UMA PROPOSTA DE AMBIENTE INTERNO PARA IMPLEMENTAÇÃO DE SGBD EM SISTEMAS DE PEQUENO PORTE E SUA UTILIZAÇÃO NA IMPLEMENTAÇÃO MICROLOBAN

Jorge Silva Dantas

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.).

Prof. Estevam Gilberto de Simone

Prof. José Lucas M. Rangel Netto

Prof. Gernot Richter

Profa. Beatriz Z. Miyasato

Rio de Janeiro, RJ - Brasil

Novembro de 1982

DANTAS, JORGE SILVA

Uma Proposta de Ambiente Interno para a Implementação de SGBD em Sistemas de Pequeno Porte e sua Utilização na Implementação MICROLOBAN |Rio de Janeiro| 1982.

VIII, 181p., 29,7 cm (COPPE/UFRJ, M.Sc., Engenharia de Sistemas e Computação, 1981).

Tese - Univ. Fed. Rio de Janeiro. Fac. Engenharia

1. Banco de Dados I. COPPE/UFRJ II. Título (Série).

Aos meus Pais

AGRADECIMENTOS

Inicialmente gostaria de agradecer à professora Beatriz Zakimi Miyasato pela competente orientação, além do incentivo, dedicação, paciência e amizade sempre demonstrada.

Ao professor Jano Moreira de Souza pela orientação inicial e apoio.

Aos professores Estevam Gilberto de Simone e Lídia Micaela Segre pela amizade e incentivo.

À Vera Lúcia D'Albuquerque pelo apoio técnico e emocional.

Aos amigos e colegas do projeto MINIBAN/COPPE, Antônio Carlos dos Santos e Antônio Cláudio Gomez de Sousa.

Aos amigos de República José Carlos de Toledo, Mário Roizman, Marcos Madeiro, Ronaldo Cabral, Reginaldo Figueiredo, Neide Maria de Andrade e todos os outros que por lá passaram.

Aos amigos Armando Augusto Clemente, José Ary Souto e Miguel Argolo.

À CAPES, CNPq e FINEP pelo apoio financeiro.

Ao Programa de Engenharia de Sistemas bem como aos seus Professores e Funcionários.

À Denise Schwartz Cupolillo pelo trabalho datilográfico e a Gilmar Fernandes pelos desenhos.

RESUMO

Este trabalho apresenta uma proposta de Ambiente Interno a ser utilizado por implementações de Interfaces de Banco de Dados em sistemas de pequeno porte e define como a implementação MICROLOBAN poderia ser dirigida a este ambiente.

Inicialmente, é preciso esclarecer que este ambiente interno não inclue apenas Estruturas de dados e métodos de acesso mas também um método de Gerência de Memória, de reconstrução dos dados, uma forma de descrever como os dados são armazenados internamente e um conjunto de rotinas que se encarregam de fazer a comunicação do nível conceitual com o interno proposto.

Em seguida são apresentadas as construções MICROLOBAN, é feito o mapeamento destas para o Ambiente proposto e são apresentadas as principais primitivas de acesso associadas à implementação MICROLOBAN de modo a realizar o mapeamento entre os níveis conceitual e interno.

ABSTRACT

This work presents a proposal of an Internal Enviroment that may be used by Data Base Interfaces implementations in mini/micro systems and it offers a way how a MICROLOBAN implementation may be oriented to this enviroment.

First it is necessary to make clear that this internal enviroment does not include only data structures and acess methods but also a data reconstrution method, a memory management and a way of describing how data are internally stored and a set of routines for communicating the conceptual and the proposed internal levels.

Finally, MICROLOBAN constructions and their mapping to the proposed enviroment are presented including the main primitives of access associated to the MICROLOBAN implementation that communicates the conceptual and internal levels.

ÍNDICE

Páginas

CAPÍTULO I

I. INTRODUÇÃO

Este trabalho faz parte de um esforço realizado pela equipe de Software de Suporte da COPPE/UFRJ no sentido de pesquisar e produzir tecnologia na área de Banco de Dados, visando principalmente a emergente indústria nacional de computadores, e é o resultado de uma das etapas do projeto MICROBAN cujo objetivo final é a implementação de um subconjunto da Linguagem de Operação de Banco de Dados (LOBAN) no computador COBRA-300.

I.1. HISTÓRICO DO PROJETO

A interface LOBAN acima citada surgiu como resultado da 1ª etapa do projeto MINIBAN - Projeto de um Sistema de Banco de Dados em Minicomputador Nacional - originário de um convênio entre o CNPq, a GMD da República Federal da Alemanha, DIGIBRÁS e UFRGS, o qual começou a ser desenvolvido em 1977.

MINIBAN tem como objetivo a especificação e desenvolvimento de um Sistema de Banco de Dados para um minicomputador nacional, assim como o desenvolvimento de tecnologia nacional na área de Banco de Dados. A primeira etapa do projeto concentrou-se na especificação das estruturas de informação, nas operações usando essas estruturas, assim como a definição de LOBAN |Richter[8],[9]|. A primeira implementação de um subcon

junto LOBAN, denominado Sistema L |Heuser[6]|, está sendo desenvolvido pela UFRGS. A partir de outubro de 1978 a COPPE/UFRJ iniciou sua participação neste projeto, dando origem ao projeto MINIBAN/COPPE.

A fase inicial do projeto MINIBAN/COPPE consistiu da definição completa e detalhada de LOBAN, com a principal característica de separar nitidamente a etapa de definição da interface Usuário/Banco de Dados, da etapa de realização ou implementação da mesma em um sistema portador |D'Albuquerque[1], Dantas[3], Santos[11-12], Pinto[8]|.

Devido à extensão e extrema complexidade da interface assim definida, e ao desejo de adaptar LOBAN a um computador de pequeno porte, a equipe do projeto MINIBAN/COPPE optou, como segunda etapa do projeto (denominado MICROBAN), pelo desenvolvimento de um protótipo capaz de suportar um subconjunto LOBAN.

O sistema portador escolhido foi o COBRA-300|COBRA[31]|, por ser uma máquina nacional e pela necessidade de software básico para sistemas desse porte.

O subconjunto LOBAN definido para implementação do primeiro protótipo é denominado MICROLOBAN.

I.2. CARACTERÍSTICAS DA INTERFACE MICROLOBAN

MICROLOBAN é uma interface cuja especificação foi realizada sem a preocupação de como suas estruturas seriam representadas internamente na máquina escolhida para implementação . Esta separação deu maior liberdade de implementação pois pode se escolher dentre um maior número de alternativas, qual a solução mais viável quanto à realização no sistema portador.

Suas estruturas de informação fazem com que se tenha uma visão global da informação, e as operações sobre estas resolvem a maioria dos problemas de tratamento da informação na área de Banco de Dados. Tentou-se dar na sua definição a possibilidade de futuras extensões sem modificar suas funções originais.

Por ser uma interface poderosa, a proposta inicial é que sua definição e implementação se preste como ferramenta de ensino na área de Banco de Dados, e que funcione como um sistema de referência na comparação das diversas abordagens.

MICROLOBAN é uma linguagem autocontida, em português, que engloba diversas funções, dentre as quais estão as que descrevem a informação e seu relacionamento e as estruturas de Base de Dados (DDL), as funções de manipulação e uso da Base de Dados (DML), e as funções normalmente realizadas por utilitários do SGBD como reconstrução de acervos, modificação das definições dos dados, etc.

O principal critério adotado na definição do subconjunto que compõe MICROLOBAN, foi o de reduzir a complexidade de LOBAN a um nível implementável no sistema portador, porém man

tendo a sua filosofia básica.

A comunicação do usuário com o Sistema de Banco de Dados é feita utilizando uma instrução de trabalho, a qual corresponde à realização de um serviço. Uma instrução de trabalho é constituída de uma instrução de início, uma lista de instruções autônomas e uma instrução de fim. Esta instrução de início determina se o processamento será interativo ou em lotes. Os dados a serem processados serão fornecidos com as instruções como anexos, ou separados em arquivos externos. Como resposta são fornecidos o resultado externo (ex.: relatórios), e o interno que é guardado como uma Base de Dados caso tenha sido feito alterações sobre a mesma. Serão também fornecidas mensagens operacionais padronizadas informando as ocorrências durante o processamento.

No que diz respeito a Entrada/Saída de Dados, será utilizada uma única máscara (formato) padrão predefinida e várias regras de interpretação e representação, respectivamente.

- PRETIPOS SUPORTADOS POR MICROLOBAN

MICROLOBAN permite alguns dos pretipos de LOBAN entre os quais podemos citar:

- Pretipos atômicos: Real, inteiros, sigla, data e hora;
- Pretipos agregados básicos: tupla, ligação, coleção de itens, tabelas relacionais e ligacionais;
- FOLHA, que conterá a descrição dos tipos de construções que formam a Base de Dados (verbetes de coe

rência), as autorizações permitidas (Verbetes de acesso), identificação de usuários (Verbete de usuário) e procedimentos predefinidos (Verbete de texto fonte);

- ARQUIVO, composto de um tipo de construção padrão chamado FICHA e uma tabela. Sendo assim existem arquivos relacionais e ligacionais. Uma restrição feita em MICROLOBAN é a eliminação das relações de ordem tanto dos arquivos quanto das ligações; e
- FICHA, contendo informações referentes ao arquivo ou acervo setorial tais como: data de criação, data da última atualização, etc ...

Os pretipos ACTRAB (Acervo de Trabalho), ACSET (Acervo Setorial) e ACTOT (Acervo Total), foram mantidos de LOBAN.

- COMANDOS MICROLOBAN

- Comando de Gerência, onde o usuário tem a possibilidade de criar e abolir tanto Acervos Setoriais quanto arquivos;
- Comandos de Manipulação, que permitem incluir, excluir e substituir construções tanto na Base de Dados quanto no Canal Auxiliar;
- Comandos de controle de fluxo, que permitem a execução repetitiva (comando condicional).
  O comando iterativo permite a especificação de uma ordenação temporária sobre o conjunto de pontos a serem processados;

- Comandos de alocação de recursos com os quais o usuário informa a Base de Dados a ser usada, assim como os dispositivos de entrada/saída necessários. Além de definir qual a Base de Dados requisitada, o usuário deverá especificar a área de dados a ser usada (protegida) e para que tipo de acesso;
- Comandos de marcação, que permitem a definição de "*snapshots*", e não de vistas como é o caso de LOBAN;
- Comandos de reconstrução. São permitidos dois níveis de reconstrução: reconstrução de sessão (de tipo regressiva) e reconstrução de comandos dentro de uma sessão (tanto progressiva quanto regressiva);
- Comando de saída que permite representar construções no meio externo;
- Comando de definição de transação, que permite definir um conjunto de comandos cuja execução será considerada como uma unidade de processamento. No início da transação são especificadas as ações que serão executadas quanto da ocorrência de erros de execução.

Além dos comandos acima descritos temos as expressões que quando executadas geram construções na Zona Intermediária (área de trabalho do sistema). Estas expressões não têm restrições quanto ao nível de embutimento e englobam as seguintes funções:

- Entrada de dados
- Operações aritméticas
- Operações da álgebra relacional
- Operações do cálculo relacional

- Operações sobre tabelas relacionais e ligacionais

MICROLOBAN não permite operações sobre cadeias de caracteres, produto e concatenação cartesiana.

Por último, temos as expressões que permitem o endereçamento de pontos da Base de Dados ou Canal Auxiliar, e campos dos volumes de entrada/saída.

I.3. ARQUITETURA GERAL DO SISTEMA

Os Canais Primários de Entrada/Saída estabelecem a comunicação entre o usuário e o sistema de Banco de Dados. Eles recebem do usuário, comandos MICROLOBAN e do sistema, mensagens de execução.

O ANALISADOR se encarrega das fases de análise léxica, sintática e semântica estática. O analisador léxico utiliza a tecnologia usual de autômatos finitos para localização e tabelas de espalhamento ("*hashing*") para reconhecimento de palavras reservadas. O analisador sintático utiliza o método RRP LL(1) |Simone[13]| com esquema de recuperação de erros por eliminação de frase. Em particular quando o modo de operação é 'interativo', o sistema elimina o comando corrente obrigando sua resubmissão a partir do ponto de erro; quando em modo "lote" o recuperador trabalha por eliminação de frases ("*panic mode*"). A escolha deste método de análise deveu-se principalmente às suas excelentes características de economia de espaço, restrição fundamental da implementação.

Para o analisador semântico optou-se por seguir a se

guinte estratégia: toda análise semântica dependente de informações contidas no esquema conceitual (FOLHA) será transferida para a fase de execução mediante a inserção dos comandos correspondentes no código interpretável. Esta solução permitiu simplificar sensivelmente o analisador semântico e se justifica por duas razões: em primeito lugar o tempo de análise, tradução e interpretação deverá ser desprezível frente ao tempo médio de execução de uma operação sobre a base de dados; em segundo lugar o modo preferencial de operação deverá ser "interativo", quando a execução se dá comando a comando, reduzindo o intervalo entre a análise e execução.

O analisador de semântica estática, assim reduzido, compõe-se apenas de uma pilha de trabalho e de algumas rotinas comandadas pelo analisador sintático. O ANALISADOR também comanda o TRADUTOR que gera código intermediário em forma de árvore na qual estão incluídos comandos de análise semântica que dependem do conteúdo da base de dados.

O INTERPRETADOR recebe do TRADUTOR uma árvore cujos nós contêm as instruções intermediárias a serem executadas pela máquina MICROLOBAN ou pelo próprio INTERPRETADOR. A árvore será percorrida em forma posfixa pelo INTERPRETADOR que comanda o EXECUTOR fornecendo uma ordem de cada vez. Alguns comandos requerem a execução repetitiva de subárvores (por exemplo, comandos iterativos, avaliação de endereço de ponto, etc) cujo controle também ficará a cargo do INTERPRETADOR.

Uma característica importante do sistema, é o fato de não ser necessária a criação de uma tabela de símbolos para o ANALISADOR, já que a análise semântica será realizada em tempo

de execução consultando o esquema conceitual da Base de Dados que é a FOLHA correspondente.

O canal REGISTRADORES é capaz de armazenar valores booleanos, definições de tipo e pretipo de construções, código intermediário e outras informações necessárias à execução do referido código, além de possuir pilhas que conterão operadores, nome, tipo, pretipo das construções geradas na Zona Intermediária e outras informações colocadas pelo INTERPRETADOR. O Sistema possui uma área de "*buffers*" na MEMÓRIA PRINCIPAL , onde serão lidas/escritas tuplas, cadeias de caracteres e textos fontes a serem gravadas na FOLHA da Base de Dados. Após a execução de um código intermediário o EXECUTOR colocará nos registradores, os resultados obtidos e que eventualmente serão usados na execução dos próximos códigos intermediários. Como podemos notar, o EXECUTOR MICROLOBAN é quem comanda todo acesso e manipulação da Base de Dados, Canal Auxiliar e Zona Intermediária; entrada e saída de dados e avaliação de expressões, ou seja, ele é responsável pela execução dos códigos intermediários fornecidos pelo INTERPRETADOR. Essa parte de tradução/interpretação, assim como os registradores e o executor MICROLOBAN é detalhadamente explicada em |D'Albuquerque[2]|.

Os Canais Secundários de Entrada/Saída são responsáveis pela comunicação do Sistema com o exterior, no que eles possibilitam a leitura/gravação de dados em dispositivos de armazenamento secundário.

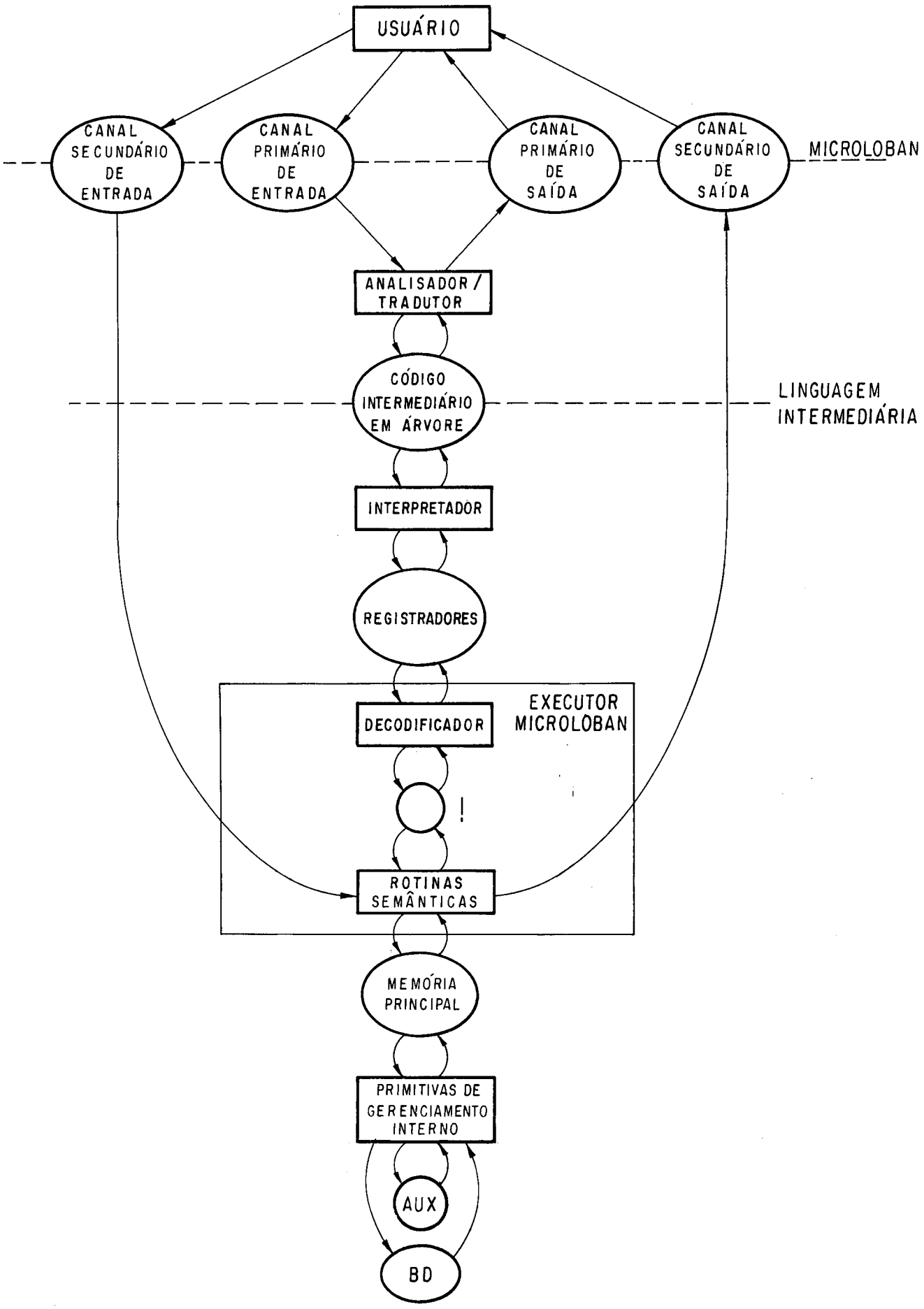


Fig. I.1 - Arquitetura Geral do Sistema

I.4. OBJETIVOS DO TRABALHO

Embora este trabalho tenha inicialmente sido planejado para propor a organização interna dos dados e primitivas de acesso a serem suportadas pela implementação MICROLOBAN, durante o seu desenvolvimento percebeu-se que com algum esforço adicional ele poderia se tornar uma proposta mais geral de um Ambiente Interno a ser adotado por outras implementações de Sistemas de Gerência de Base de Dados em máquinas de pequeno porte.

A partir daí então nós optamos pelos seguintes objetivos maiores para este trabalho:

a) Apresentar uma proposta de um Ambiente Interno geral que aborda não apenas as estruturas de dados e métodos de acesso a serem suportados a nível interno, mas também a Gerência da Memória, um método de Reconstrução dos dados armazenados, uma forma de descrição (Esquema Interno) dos dados armazenados e um conjunto de primitivas de acesso que servirá como interface entre os níveis conceitual e interno da implementação; e

b) Realizar o mapeamento entre as construções MICROLOBAN e o Ambiente Interno proposto, de modo a que este possa ser adotado pela implementação MICROLOBAN.

A figura I.2 ilustra os níveis de abstração envolvidos na implementação MICROLOBAN, a partir dos objetivos acima citados.

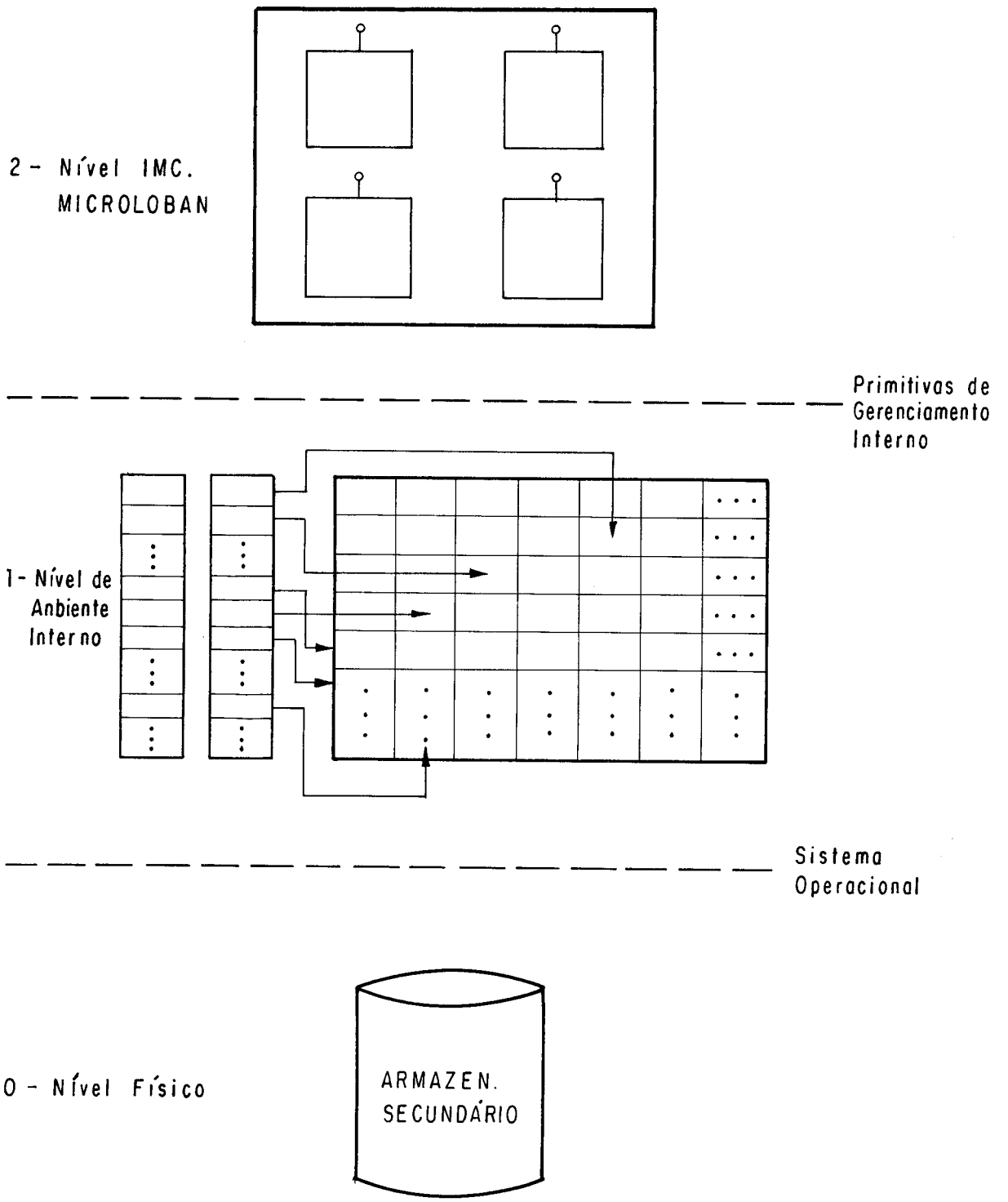


Fig. I.2 - Níveis de Abstração da Implementação MICROLOBAN

I.5. ORGANIZAÇÃO DA TESE

O presente trabalho encontra-se organizado em capítulos, cada um com uma função bem definida porém com um sequenciamento lógico que visa atingir os objetivos anteriormente listados.

O capítulo II apresenta a proposta de Ambiente Interno anteriormente falada, de uma maneira mais ou menos desvinculada da implementação alvo.

O capítulo III apresenta rapidamente as construções MICROLOBAN de modo a dar uma noção de como a interface vê os dados e quais as estruturas de dados a serem internamente representadas. Detalhes maiores poderão ser encontrados em |Santos[12]| e |Richter[8]|

O capítulo IV se encarrega de fazer a ligação entre o II e o III, no que realiza o mapeamento entre as estruturas de dados conceituais (construções) e as internas apresentadas na proposta geral.

O capítulo V descreve como seria organizado o Esquema Interno MICROLOBAN, o qual descreveria não apenas a representação interna das construções, mas também algumas informações suplementares mantidas pelo SGBD de forma a poder executar a sua tarefa de controle geral do Sistema de Banco de Dados.

Finalmente o capítulo VI apresenta as conclusões chegadas através do desenvolvimento do trabalho e suas próximas etapas.

Além dos capítulos acima citados, um Anexo também é apresentado, o qual contém algoritmos para as principais pri

mitivas que se encarregariam de realizar a comunicação entre as rotinas semânticas e o nível interno da implementação MICROLOBAN, as quais segundo a arquitetura apresentada na figura I.1 foram chamadas de *primitivas de Gerenciamento Interno*.

CAPÍTULO II

II. PROPOSTA DE UM AMBIENTE INTERNO PARA A IMPLEMENTAÇÃO DE INTERFACES DE BANCO DE DADOS EM SISTEMAS DE PEQUENO PORTE

Na tentativa de se especificar um esquema de armazenamento interno para a implementação MICROLOBAN verificou-se que o definido, com alguns ajustes, poderia se tornar geral de forma a poder ser adaptado a qualquer interface relacional a ser implementada em um sistema de pequeno porte, ou até mesmo a interfaces não relacionais como é o caso de MICROLOBAN. Daí então optou-se como norma de implementação, pela definição de um esquema que fosse eficiente no armazenamento das construções MICROLOBAN, porém contendo uma estrutura própria independente destas, de forma a servir como uma ferramenta geral na implementação de SGBDs. Com isso alcançou-se uma grande uniformidade no armazenamento físico das construções MICROLOBAN, uma grande independência do nível interno em relação ao conceitual |Date[22]| e uma maior clareza na separação das rotinas em semânticas e primitivas.

Neste capítulo nós trazemos não apenas uma proposta de Estruturas de Dados a serem suportadas pela implementação a nível físico, mas também os métodos de acesso associados, como poderia ser feito o gerenciamento da parte da memória principal alocada para *"buffers"*, um método de registrar as alterações feitas na Base de Dados Armazenada de modo a poder reconstrui-la no caso de alguma falha ocorrer, como os dados armaze

nados encontram-se descritos de modo a serem corretamente manipulados pelo SGBD e as primitivas que manipulam as informações a esse nível (interno), as quais na arquitetura apresentado no capítulo I, figuta I.1, foram chamadas de *primitivas de Gerenciamento Interno*. Daí a termos chamado de proposta de um Ambiente Interno.

Como usado aqui, o termo Base de Dados Armazenada, ou BDA como aparecerá muitas vezes, engloba todos os dados realmente armazenados, ou seja, não apenas os dados de interesse dos usuários, mas também as informações de controle necessárias ao funcionamento do SGBD.

De forma a simplificar a explicação e facilitar a compreensão da proposta, nós adotaremos o modelo relacional como o escolhido para implementação através do ambiente interno que estamos definindo, de forma a poder usar a terminologia relacional (tuplas, relações, etc), porém advertindo que o mesmo também pode ser adotado por outros tipos de abordagens, como mostraremos nos capítulos seguintes para MICROLOBAN, desde que pequenas adaptações sejam promovidas.

II.1. DISTRIBUIÇÃO FÍSICA DOS DADOS

Embora este capítulo apresente uma proposta geral de um Ambiente Interno para máquinas de pequeno porte, nesta seção frequentemente nos referiremos especificamente ao COBRA-300 |COBRA[31]| (máquina onde será feita a implementação MICROLOBAN), devido à necessidade de se fixar detalhes fortemente associados à máquina alvo, visando principalmente simplificar o entendimento da proposta. Assim se a máquina escolhida para aplicação da proposta não for esta, pequenas adaptações deverão ser feitas principalmente nos aspectos mais próximos à máquina, tais como: tamanho do setor no dispositivo de armazenamento secundário, tamanho do registro, capacidade de armazenamento, tamanho de ponteiros, etc.

Dependendo do software de manipulação de arquivos disponível no sistema operacional da máquina alvo, pode ser que se possa usar uma das organizações de arquivo por este suportadas, livrando-se assim a implementação da carga do controle do armazenamento secundário; ou pode ser que seja necessário que a própria implementação se encarregue disso, de forma a garantir padrões mínimos de eficiência. A proposta que apresentamos baseia-se na adoção de um único arquivo relativo para armazenar todos os dados presentes à BDA, principalmente devido à restrição, muito comum nos sistemas de pequeno porte, ao número de arquivos que podem estar abertos ao mesmo tempo, à economia de espaço e ao acesso direto inerentes à organização relativa.

Dessa forma, a implementação MICR@LOBAN adotará um único arquivo relativo SOM (sistema operacional do COBRA-300), para armazenar toda a BDA. Devido à limitação SOM de um arquivo não poder ocupar mais de um volume, uma outra restrição é colocada no armazenamento da BDA, que é não poder ocupar mais de um disquete.

A Base de Dados Armazenada (BDA) será constituida de páginas de 255 bytes e ocupará um disquete inteiro (1024 páginas). Assim o arquivo relativo usado para armazená-la será dividido em registros de 255 bytes, de forma a estabelecer uma correspondência página-registro. A escolha do tamanho 255 para a página, foi feita em virtude do setor do disquete ser constituido de 255 bytes e do software do relativo reservar um byte por registro para controle próprio. Assim cada registro relativo ocupará exatamente um setor.

Com a divisão do armazenamento físico disponível para a BDA em páginas de tamanho fixo (255 bytes), este passa a ser assumido como o tamanho padrão para todas as operações de leitura e gravação no disco, ou seja, a página passa a ser a unidade básica de recuperação/gravação do ponto de vista das rotinas que lidam diretamente com o armazenamento físico (*primitivas de Gerenciamento Interno*).

O formato de cada uma das páginas que compõem a BDA é mostrado na figura II.1.

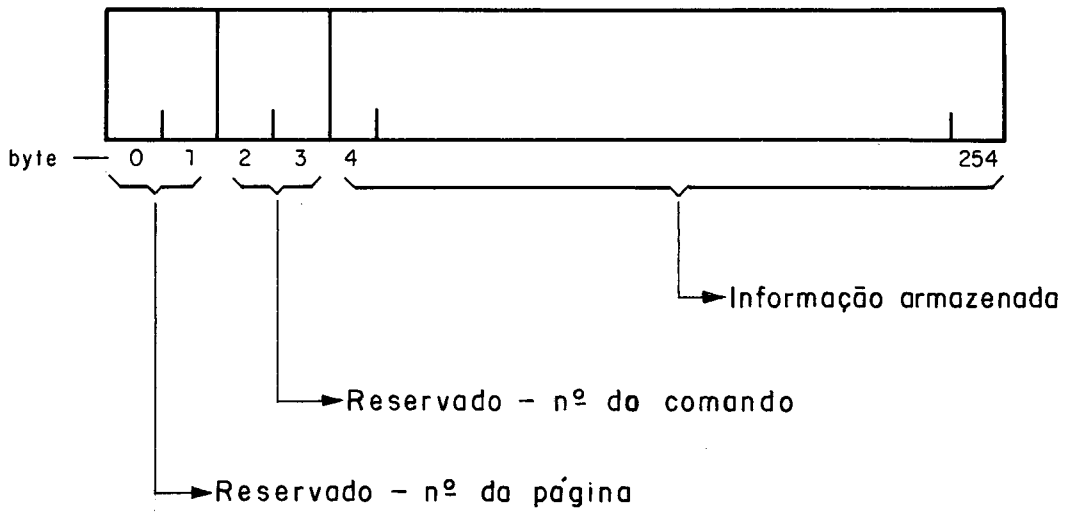


Figura II.1 - Formato de uma página no disquete.

Como pode-se ver na figura II.1, os quatro primeiros bytes de cada página são reservados para guardar informações de controle, necessárias ao funcionamento do sistema. O valor do campo "nº de página", informa qual o número (posição relativa do registro correspondente no arquivo SOM) da página em jogo. Esse campo é útil para a alocação de páginas, para a gerência da área da memória principal reservada para a paginação e para a "reconstrução da BDA", como será visto ainda neste capítulo. O valor do campo "nº de comando", é reservado para informar o número do comando que alterou a página em jogo, de forma a possibilitar a reconstrução da BDA até o estado que esta se encontra após a execução de um determinado comando.

As páginas constituindo o arquivo SOM é que se encarregarão de guardar as tuplas pertencentes às relações armazenadas, cada uma recebendo tantas tuplas inteiras quanto possível. Deve-se observar contudo, que uma página é característi

ca de uma relação, no que ela só contém tuplas de uma única relação, sendo que estas caracterizam-se por ser do mesmo tipo e tamanho.

- ALOCAÇÃO DE PÁGINAS

A alocação dinâmica do espaço disponível é um dos prérequisitos básicos a serem satisfeitos por implementações como esta que estamos descrevendo, uma vez que a capacidade de armazenamento físico fornecida pelo sistema, geralmente é muito pequena. Daí a introdução de procedimentos que se encarregam de alocar páginas apenas quando indispensável e de as liberar imediatamente após o momento que as mesmas deixam de ser necessárias.

De maneira a controlar a alocação de páginas, uma pilha será mantida, a qual conterá a cada momento os números de todas as páginas não alocadas, e portanto passíveis de alocação. No início (criação da Base de Dados), a pilha deverá ser preenchida com valores decrescentes dos números de todas as páginas disponíveis no armazenamento secundário (1024 → 1 na nossa implementação), de maneira a que as de número mais baixo (início do disquete) sejam as primeiras a serem alocadas. À medida que se vai introduzindo e retirando dados, páginas vão sendo alocadas e liberadas e seus números vão sendo retirados ou colocados no topo da pilha, de forma a que a última página liberada, seja sempre a primeira a ser alocada. Assim, a pilha cresce ou diminui em sentido inverso ao da BDA, ou seja, se a pilha está vazia, a BDA está cheia e vice-versa.

A forma como essa pilha (chamada de "Pilha de Páginas Disponíveis") será armazenada, encontra-se descrita na seção II.4.

II.2. ORGANIZAÇÃO LÓGICA DOS DADOS

Como foi dito na seção anterior, as páginas que compõem a BDA encontram-se distribuidas por todo o armazenamento físico. É através do grupamento lógico dessas páginas, que chegamos às construções internas que representarão todas as construções da Base de Dados. Essas construções internas são chamadas tabelas básicas e os grupamentos lógicos que as formam são realizados através da utilização de um índice ao qual damos o nome de Índice de Páginas, uma vez que a sua função é relacionar as páginas que compõem as tabelas básicas.

Em virtude do Índice de Páginas ser único, é necessário que seja blocado de maneira a formar todas as tabelas básicas presentes à BDA. A figura II.2 mostra como as tabelas básicas são formadas a partir do Índice de Páginas.

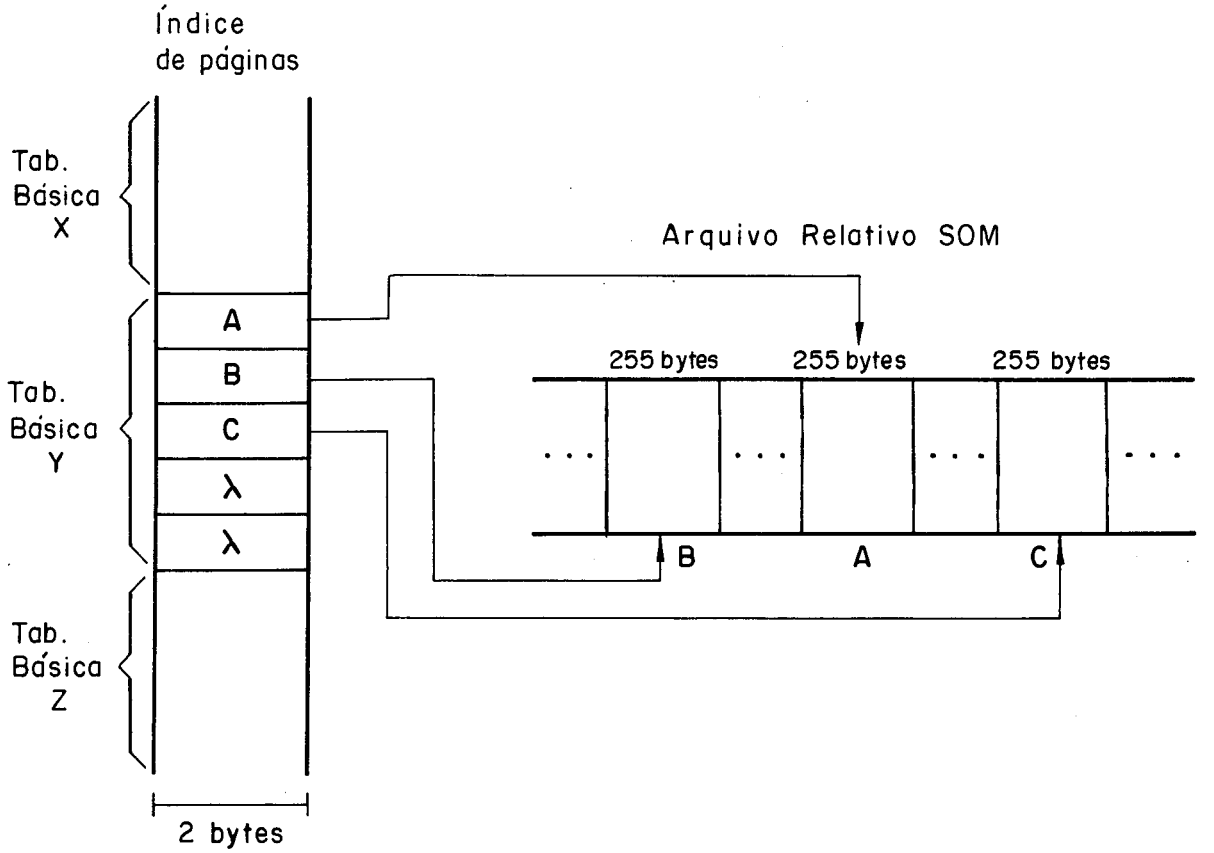


Figura II.2 - Esquema de formação de tabelas básicas.

Como pode ser visto na figura II.2, no que diz respeito à sua organização lógica, uma tabela básica é constituida de uma série de entradas no Índice de Páginas e de um conjunto de páginas (contendo tuplas) no armazenamento físico, sendo que cada uma das entradas aponta para uma das páginas ou possui o valor vazio (λ) significando que a página correspondente não se encontra alocada. A cada uma das entradas dá-se o nome de página lógica, sendo página física a sua correspondente (se houver) no arquivo SOM.

Em virtude dos conceitos acima apresentados e uma vez que toda a proposta gira em torno das tabelas básicas, listamos as suas principais diferenças e características, de forma a fa

cilitar a compreensão.

As páginas lógicas diferem das físicas, basicamente nos seguintes pontos:

a) Quanto ao tamanho.

Devido ao fato de conter apenas um ponteiro, uma página lógica é composta de 2 bytes, enquanto que uma física tem 255 bytes que é o tamanho do registro que a armazena. Os 2 bytes reservados para ponteiro são mais do que suficientes em nossa implementação, uma vez que temos apenas 1024 páginas (número máximo de registros em um arquivo relativo SOM) para serem apontadas.

b) Quanto à alocação.

Quando uma tabela básica é criada, são alocadas tantas entradas para ela no Índice de Páginas quanto for o número máximo de páginas físicas que ela poderá vir a ter. Esse número é função da cardinalidade máxima, e tamanho da tupla especificados para a relação a ser armazenada pela tabela básica.

À medida que se vai incluindo tuplas na relação, páginas físicas vão sendo preenchidas, e novas páginas físicas vão sendo alocadas assim que as já alocadas vão ficando completas.

Dessa maneira, conlui-se que devido a serem alocadas todas de uma vez, as páginas lógicas são consideradas como de alocação estática, ao contrário das físicas que o são apenas quando necessário, caracterizando uma alocação dinâmi

ca. A alocação dinâmica das páginas físicas é uma política indispensável em implementações como esta que estamos descrevendo, devido ao pequeno porte do sistema portador.

c) Quanto ao grupamento.

Como pode-se ver na figura II.2, enquanto as páginas lógicas de uma tabela básica estão fisicamente grupadas, as físicas encontram-se distribuidas aleatoriamente no armazenamento secundário. Esta é mais uma consequência da adoção de uma política de alocação dinâmica das páginas físicas.

De maneira a possibilitar o acesso direto às tuplas através do conhecimento das posições relativas que elas ocupam na tabela básica e de forma a garantir uma ocupação ótima das páginas físicas alocadas para esta, as seguintes políticas deverão ser mantidas pela implementação:

a) Compactação das páginas lógicas ocupadas de uma tabela básica.

As páginas lógicas ocupadas de uma tabela básica deverão estar sempre compactadas no início da área blocada para ela no Índice de Páginas, ou seja, não pode haver nenhuma com valor vazio entre elas. Assim quando da alocação e liberação de páginas físicas, o SGBD se encarrega de manter a compactação entre as páginas lógicas associadas.

b) Compactação das tuplas nas páginas.

As tuplas que compõem uma tabela básica sempre encontram-se compactadas, seja dentro da página física que as contêm, seja dentro da tabela básica como um todo. Para que isso seja alcançado, a seguinte política de atualização deverá ser mantida para as tabelas básicas.

. a inserção de uma tupla em uma tabela básica sempre será feita na última página física (apontada pela última página lógica ocupada) alocada para esta tabela, logo após a última tupla presente nesta página; a não ser no caso desta já estar cheia o que implicará na alocação de mais uma página física para esta tabela básica e na inserção da tupla pretendida nesta nova página.

. a remoção de uma tupla de qualquer uma das páginas implicará no preenchimento do buraco deixado, pela última tupla (última tupla presente na última página alocada) da tabela básica, a não ser no caso de ser esta (a última) a removida.

A figura II.3, esquematiza a formação de uma tabela básica levando em conta as regras acima descritas.

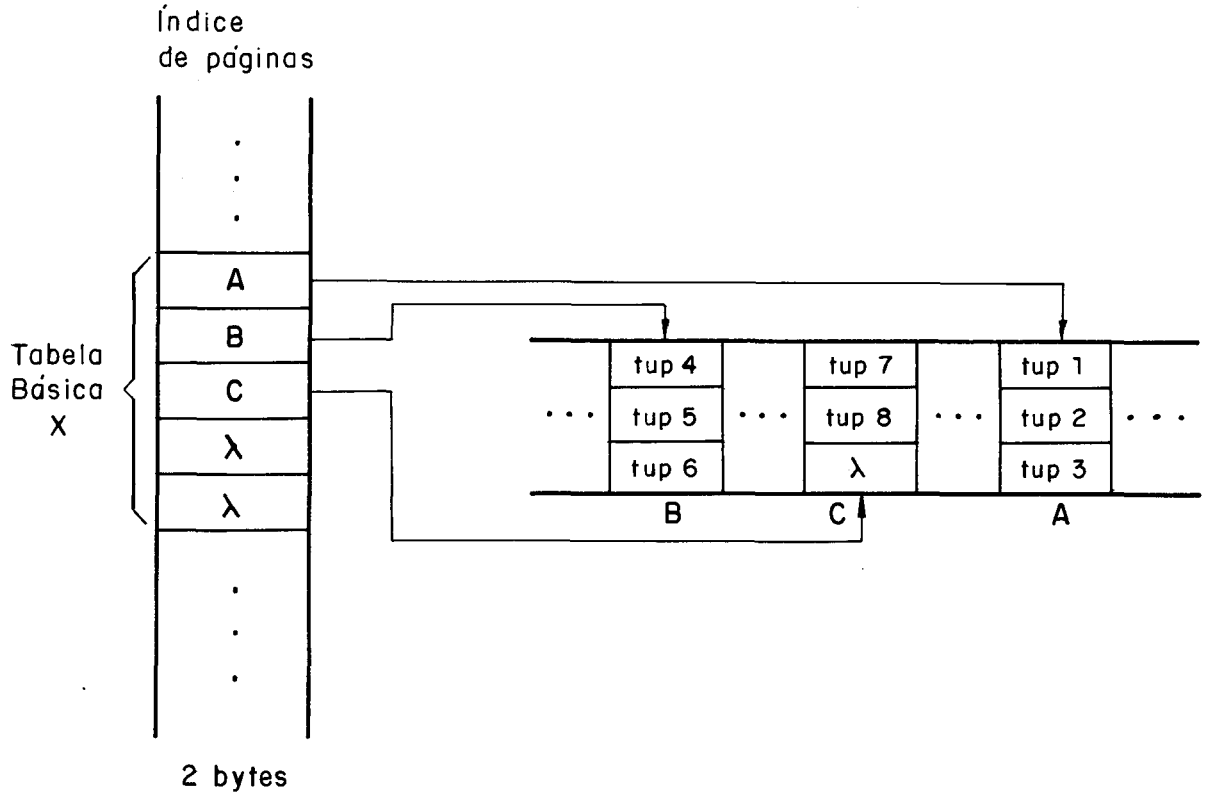


Figura II.3 - Esquematização completa de uma tabela básica.

Observações:

1 - A tabela foi criada e a única operação realizada nela até agora, foi a de inclusão. A ordem de inclusão das tuplas foi: tupl, tup2, tup3, ... tup8;

2 - Cada página física comporta apenas 3 tuplas básicas inteiras;

3 - Se a próxima operação for uma inclusão, a tupla será incluída na página física C, no "buraco" com λ (vazio). Se após esta, fosse feita outra inclusão, uma nova página física teria que ser alocada para a tabela básica;

4 - Se a próxima operação for uma retirada, a tupla "tup8" seria deslocada para o buraco deixado pela removida, a não ser no caso de ser ela ("tup8") a removida.

II.3. MÉTODOS DE ACESSO ÀS TUPLAS NAS TABELAS BÁSICAS

O método de acesso às tuplas inerente a uma tabela básica é o sequencial-indexado. Para cada uma das páginas lógicas ocupadas, acessa-se a página física correspondente e procura-se a tupla desejada dentro desta. Como pode-se ver, esse método na verdade implica em uma busca sequencial.

Devido à ineficiência do método acima citado, alguns artifícios são empregados no sentido de possibilitar um acesso mais direto às tuplas de uma tabela básica.

Um destes artifícios, já falado na seção anterior, é a manutenção da compactação entre as tuplas de uma tabela básica. Esse procedimento, além de possibilitar um melhor aproveitamento do espaço já que elimina os buracos vazios entre as tuplas, também capacita o acesso "direto" a estas de acordo com a posição relativa que elas ocupam na tabela básica. Como as tuplas estão compactadas, sabendo-se o seu tamanho e posição relativa, calcula-se em que página lógica ela está e com qual deslocamento, e acessa-se a entrada correspondente no Índice de Páginas, obtendo-se o número da página física onde a tupla pretendida se encontra.

Uma variação do método acima, acontece quando conhece se antecipadamente a página lógica e o deslocamento da tupla desejada. Assim, o cálculo acima citado não necessita ser realizado e o acesso é feito da mesma forma que no método anterior, ou seja, conhecendo-se a página lógica acessa-se o Índice de Páginas de modo a obter o número da página física correspondente e emite-se uma ordem de leitura do registro relativo

que a armazena. Como o deslocamento da tupla em relação ao início da página é conhecido, sabe-se exatamente onde está a tupla desejada. Esse método leva em consideração o que chamamos de TID ("*tuple identifier*"), que para nós é constituido de 2 bytes; um para a página lógica onde a tupla se encontra e outra para informar o deslocamento da tupla em relação ao início da página.

Na verdade o TID (<pag. lógica, deslocamento>) nada mais é que o *endereço lógico* de uma tupla em uma tabela básica, na medida que identifica univocamente cada tupla desta tabela básica. Dessa forma, este conceito será bastante utilizado, uma vez que é uma forma bastante eficiente e direta de se endereçar tuplas.

Contudo, um cuidado especial deverá ser tomado com tal tipo de endereçamento, uma vez que devido à mudança de posição sofrida pelas tuplas no processo de compactação, o sistema deverá se encarregar de atualizar os endereços das tuplas deslocadas pela compactação, de maneira a não incorrer em erro de endereçamento.

Um outro artifício é o que possibilita o acesso direto às tuplas, de acordo com valores conhecidos para o atributo considerado chave primária da tabela básica. As informações a respeito da chave primária para uma tabela básica são obtidas a partir da estrutura de dados conceitual que está sendo através dela armazenada, e fazem parte da descrição desta tabela básica, como será visto na seção II.4.

De maneira a possibilitar o acesso por chave, o sistema fará uso de um Índice de Espalhamento, como explicado a seguir.

- ÍNDICE DE ESPALHAMENTO

Com o objetivo de dinamizar a busca de tuplas de acordo com valores conhecidos para a chave primária, índices serão construidos sobre as tabelas básicas de acordo com transformações feitas nos valores apresentados para o(s) atributo(s) chave primária de forma a transformá-los em endereços (método de "*hashing*"). Para isso, o sistema fará uso de um Índice de Espalhamento.

Como o Índice de Páginas, o de Espalhamento também será blocado de maneira a ser usado por várias tabelas básicas. O número de entradas a alocar para uma tabela básica é função da cardinalidade máxima especificada para a estrutura de dados conceitual associada, e deverá fazer um balanceamento entre a ocupação de espaço e o número médio esperado de buscas de modo a que se obtenha valores razoáveis tanto no espaço físico gasto, como no tempo médio esperado para busca de uma tupla.

Cada entrada ocupada no Índice de Espalhamento, aponta para a tupla correspondente, através de seu TID.

A figura II.4 esquematiza a utilização do Índice de Espalhamento.

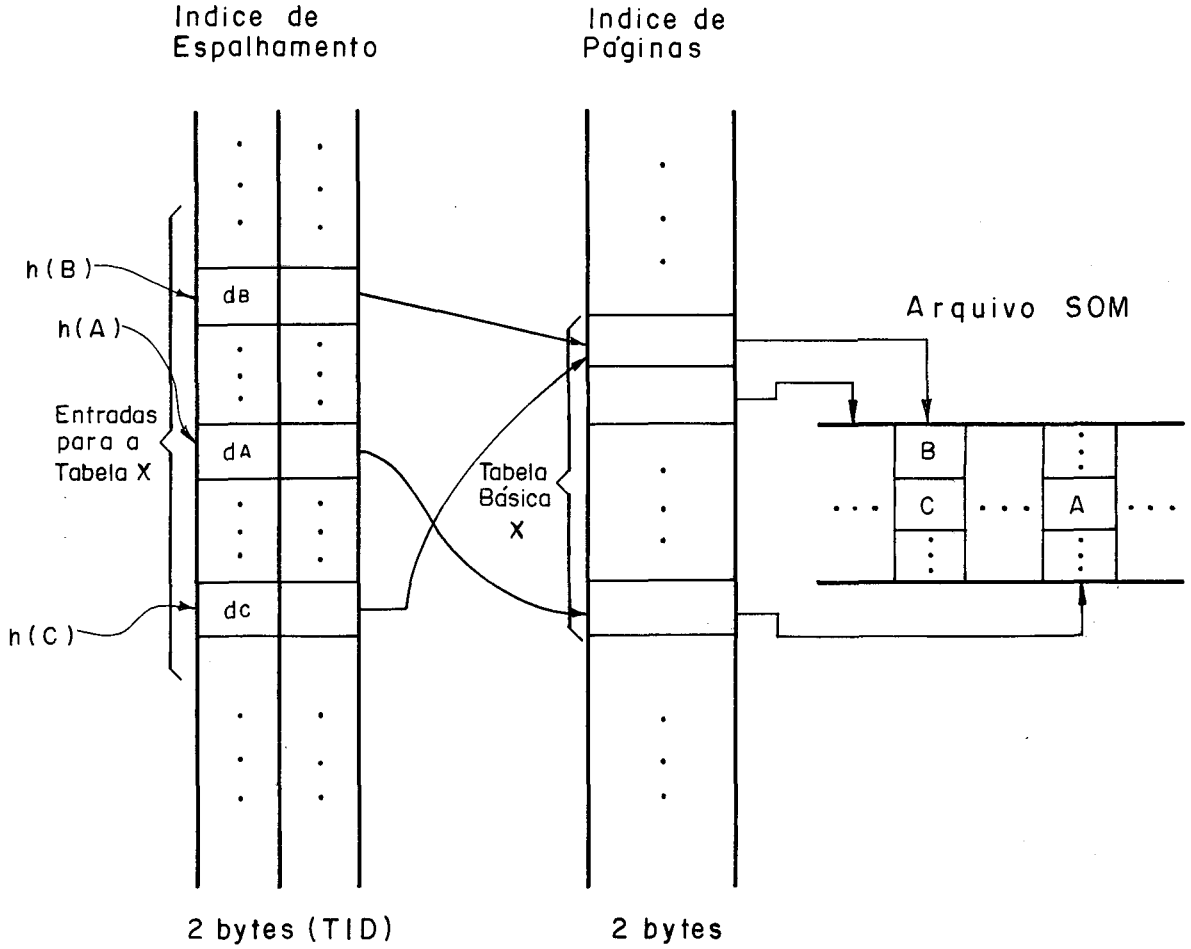


Figura II.4 - Exemplo de uso do Índice de Espalhamento.

Observações:

1 - dA, dB e dC, correspondem aos deslocamentos das tuplas A, B e C, respectivamente em relação ao início das páginas onde as mesmas se encontram;

2 - O espaço blocado para uma tabela básica no Índice de Espalhamento não será mantido compactado como no Índice de Páginas, uma vez que as entradas ocupadas dependerão das transformações nas chaves.

A transformação da chave em endereço, será conseguida através do seguinte procedimento:

a) Realizar um "*dobramento*" no valor apresentado para a chave primária, de forma a reduzí-lo a apenas um byte. Isso é feito através da divisão da chave em pedaços de 1 byte e superpondo estes através de "ou-exclusivo" ou "soma".

b) Utilizar o método de hashing duplo com quociente quadrático de forma a transformar o valor obtido em a, em uma entrada válida na área blocada para a tabela no Índice de Espalhamento.

$$h_1(K_i) = K_i \bmod n$$

$$h_j(K_i) = [h_1(K_i) + a(j-1) + b(K_i)(j-1)^2] \bmod n, \text{ para } j > 1$$

onde a é constante e

$$b(K_i) = [K_i/n] \bmod n$$

Para o tratamento de colisões, escolheu-se uma variação de um algoritmo que combina a idéia de "hashing limitado" |Clapson[21]| com a de "rearranjos" |Brent[20]| no índice, o qual foi proposto, programado e muito bem avaliado em |Souza[15]|, como Alg. 13. A idéia central deste algoritmo é só tentar rearranjar o "Índice de Espalhamento", na hora em que uma chave não puder ser incluida no limite fixado, e se contentando com a primeira troca que tornar a inserção possível.

De maneira a melhorar um pouco o tempo gasto para uma "busca sem sucesso" quando a tabela estiver com uma ocupação

baixa, resolvemos adotar a idéia do "limite dinâmico" proposto também em |Souza[15]|, porém com uma pequena variação. A idéia do limite dinâmico é começar com um limite baixo, e ir aumentando à medida que não se puder mais incluir neste limite, até um máximo pré-fixado. |Souza[15]| programou uma adaptação do limite dinâmico ao Alg. 13 e chamou de Alg. 18. O por nós adotado, contudo, diferirá um pouco deste no que o limite será dinâmico apenas na inserção, não sendo afetado pela remoção, como acontece no Alg. 18. O motivo de termos adotado essa variação do Alg. 18, é que este de modo a diminuir o limite devido à remoção, tem que registrar o número de chaves inseridas para cada limite, o que complica um pouco a gerência além de implicar em espaço adicional para contadores. Assim, cada tabela básica terá um "limite atual" associado, e esse irá crescendo à medida que não se possa incluir chaves com ele, até um limite máximo pré-fixado. O "limite atual" para cada tabela básica ficará registrado na sua descrição (Seção II.4).

Uma outra idéia para melhorar o tempo de busca de uma chave é concentrar sinônimos em uma mesma página física de modo a diminuir o número de acessos a disco. Como as tuplas sempre estão compactadas em uma tabela básica, seria necessário que se fizesse algumas trocas de modo a concentrar os sinônimos na mesma página. O problema deste esquema é que com as trocas, a inclusão pode se tornar muito lenta, em virtude da manutenção que estas acarretam, além da alteração de duas páginas físicas, o que pode sobrecarregar muito a cadeia de reconstrução (Seção II.6), já que teriam de ser registradas duas alterações ao invés de uma. A adoção de tal tipo de política, fica

assim adiada para a implementação física, em função de testes de desempenho a serem realizados.

II.4. DESCRIÇÃO DOS DADOS QUE COMPÕEM A BDA

Uma vez apresentadas as estruturas que suportarão a BDA, surge a questão de como o SGBD guarda as informações sobre os dados armazenados, de forma a manipulá-los corretamente.

Como nós já vimos é através das tabelas básicas que toda a BDA será armazenada. Assim, torna-se necessário descrever todas as tabelas presentes ao sistema, de forma a que essas possam ser corretamente manipuladas, e isso será feito através do Diretório de Tabelas Básicas, como mostrado na figura II.5.

NOME	CARDINALIDADE CORRENTE	CARDINALIDADE MÁXIMA	TAMANHO DA TUPLA	1-ENTRADA NO INDPAG	1-ENTRADA NO INDESP	DESLOCAM. DA CHAVE	TAMANHO DA CHAVE	LIMITE ATUAL

Figura II.5 - Esquematização de Diretório de Tabelas Básicas.

Todas as informações contidas nesta tabela, ou são deduzidas da construção que está sendo armazenada, ou são originárias do processamento (ex.: cardinalidade corrente), e dispensam comentários mais detalhados uma vez que os nomes são auto-explicativos e já foram citados nas seções passadas.

Depois de apresentada a forma como são descritas as tabelas básicas que compõem a Base de Dados Armazenada, fica faltando apenas a especificação de como são representadas e acessadas as estruturas de dados empregadas para suportar o esquema de armazenamento/recuperação proposto, isto é, a Pilha de Páginas Disponíveis, o Índice de Páginas, o Índice de Espalhamento e o próprio Diretório de Tabelas Básicas, as quais daqui em diante serão referidas como o *Núcleo Básico* uma vez que é em cima destas estruturas de dados que todo o sistema é montado.

De maneira a manter a uniformidade no armazenamento e recuperação, também as estruturas do *Núcleo Básico* serão representadas fisicamente em forma de tabelas básicas, exatamente como as relações da Base de Dados. Com isso consegue-se não apenas que estas sejam armazenadas e descritas juntas e da mesma forma, como também que sejam acessadas e manipuladas pelas mesmas primitivas e principalmente que também sejam reconstruidas (seção II.6) quando a Base de Dados o é.

Devido ao fato de termos estabelecido que as estruturas de dados do *Núcleo Básico* também serão representadas em forma de tabelas básicas, algumas situações particulares ocorrem. São as seguintes essas situações:

a) O Diretório de Tabelas Básicas se auto-descreve;

b) O Índice de Páginas forma a si próprio;

c) O Índice de Páginas apesar de formar a tabela básica que armazena o Diretório de Tabelas Básicas também é por este descrito;

d) O Índice de Espalhamento apesar de ser descrito pelo Diretório de Tabelas Básicas, possibilita o acesso por chave (NOME) a este; e

e) A Pilha de Páginas Disponíveis apesar de descrita pelo Diretório de Tabelas Básicas e formada pelo Índice de Páginas, deve estar avisada das páginas físicas por estas e por ela mesma consumidas, de forma a não considerá-las como disponíveis.

De maneira a evitar os "*deadlocks*" causados pelos relacionamentos acima listados, quando uma Base de Dados é criada, uma inicialização ("*bootstrap*") é feita no Índice de Páginas, no Diretório de Tabelas Básicas, no Índice de Espalhamento e na Pilha de Páginas Disponíveis, de forma a já trazer esses relacionamentos registrados. Na verdade o que a inicialização faz é montar o *Núcleo Básico* de forma que tudo mais possa ser em cima deste construido.

As figuras II.6 e II.7 mostram diagramaticamente como ficam o Índice de Páginas e o Diretório de Tabelas Básicas respectivamente, após a inicialização citada. A Pilha de Páginas Disponíveis fica preenchida por todas as páginas físicas não alocadas nesse procedimento de inicialização e o Índice de Espalhamento é inicializado a partir do seu início para apontar

para as quatro tuplas preenchidas no Diretório de Tabelas Básicas, de acordo com o método de "*hashing*" já descrito.

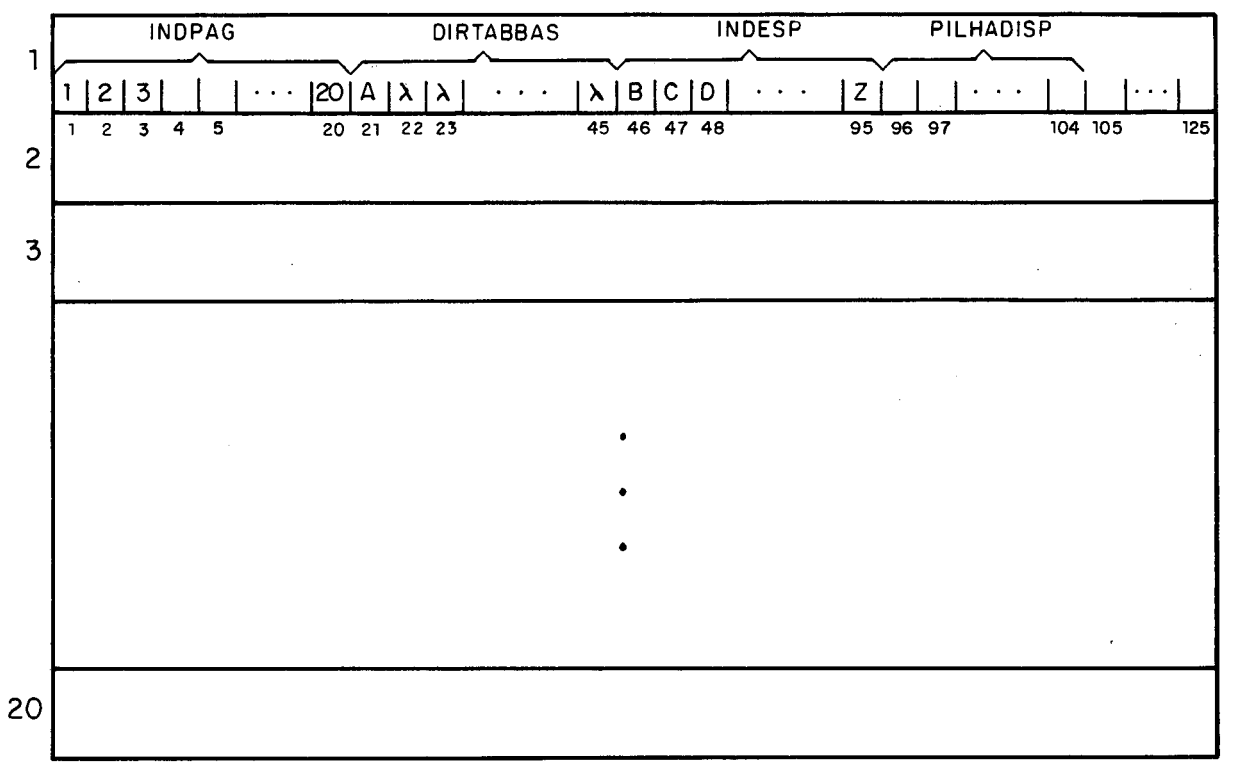


Figura II.6 - Esquematização da Organização do Índice de Páginas após a inicialização ("*bootstrap*")

Observação:

1 - As vinte primeiras páginas do disco foram alocadas para o Índice de Páginas e são apontadas pelas vinte primeiras páginas lógicas do próprio Índice de Páginas;

2 - Logo após a área blocada para o Índice de Páginas (INDPAG) nele próprio, começa a área blocada para o Diretório de Tabelas Básicas (DIRTABBAS), depois o Índice de Espalhamento (INDESP) e finalmente a Pilha de Disponíveis (PILHA

DISP), fechando as estruturas do "*Núcleo Básico*";

3 - O número de páginas lógicas alocadas para cada uma das tabelas básicas mostradas é estimada apenas como exemplo.

NOME DA TABELA	CARDINAL CORRENTE	CARDINAL MÁXIMA	TAMANHO DA TUPLA	1- ENTR. NO INDPAG	1- ENTR. NO INDESP	DESLOC. DA CHAVE	TAMANHO DA CHAVE	LIMITE ATUAL
INDPAG	104	2.500	2 bytes	1	—	—	—	—
DIRTABBAS	4	100	23 bytes	21	1	1	11 bytes	?
INDESP	?	12.500	2 bytes	46	—	—	—	—
PILHADISP	?	1.024	2 bytes	96	—	—	—	—
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
11 bytes	2 bytes	2 bytes	1 byte	2 bytes	2 bytes	1 byte	1 byte	1 byte

Figura II.7 - Esquematização do Diretório de Tabelas Básicas após a inicialização.

Observações:

1 - Mais uma vez os números foram estimados apenas para servir de exemplo, porém a partir dos estimados na figura anterior;

2 - Os valores do campo "1ª entrada em INDPAG" podem ser checados na figura anterior;

3 - A forma de relação foi empregada de forma a tornar mais fácil o entendimento e frisar que apesar de uma estrutura do *Núcleo Básico*, esta será tratada exatamente como as da Base de Dados.

O Diretório de Tabelas Básicas é o responsável pelo acesso à todas as tabelas básicas que compõem a Base de Dados Armazenada, inclusive a ele próprio.

As rotinas responsáveis pelo acesso e manipulação das tabelas básicas encontram-se descritas no fim deste capítulo e programadas no Anexo.

II.5. GERÊNCIA DA MEMÓRIA

Apesar de ficar no armazenamento secundário em virtude de não caber na memória principal, os dados que compõem a Base de Dados Armazenada necessitam ser trazidos para esta, já que é aí que o SGBD atua diretamente sobre eles. Assim, sempre que precisar de um dado o SGBD deverá descobrir a página física onde este se encontra e pedir que seja trazida para a memória principal de forma a poder atuar sobre ela.

Ainda devido à pouca disponibilidade de espaço também é necessário que se tenha uma boa gerência da parte da memória principal reservada para guardar as páginas físicas trazidas do armazenamento secundário, de forma a tirar o máximo proveito delas. É a organização desta área e a sua gerência, que apresentamos nesta seção.

- ÁREA DE PAGINAÇÃO

É a parte da memória principal responsável pelo armazenamento das páginas físicas trazidas do armazenamento secundário, além das informações necessárias à sua própria gerência.

Como pode ser visto na figura II.8 a Área de Paginação está dividida em duas partes principais:

a) Memória de Páginas (MEMÓRIA).

É responsável por receber as páginas físicas trazidas do armazenamento secundário.

A memória de páginas está dividida logicamente em entradas de 255 bytes, que é o tamanho da página física a ser guardada. Os campos NUPAG, NUCOM e INFORMAÇÃO já foram explicados na seção II.1, porém a explicação será repetida de forma a ajudar a compreensão. NUPAG informa o número da página física que está guardada na entrada em jogo, ou seja, o número do registro relativo para aí lido. O campo NUCOM, é reservado para guardar o número do comando que alterou a página em jogo. Essa informação é muito útil para a reconstrução da Base de Dados. O campo INFORMAÇÃO tem a função de armazenar as tuplas guardadas nas páginas físicas.

Uma última observação com relação à memória de páginas, é a respeito do número (N) de entradas que esta deverá possuir. Em virtude deste ser um fator altamente dependente do tamanho da memória disponível e da requisição total de espaço, este número só poderá ser fixado na fase de implementação física.

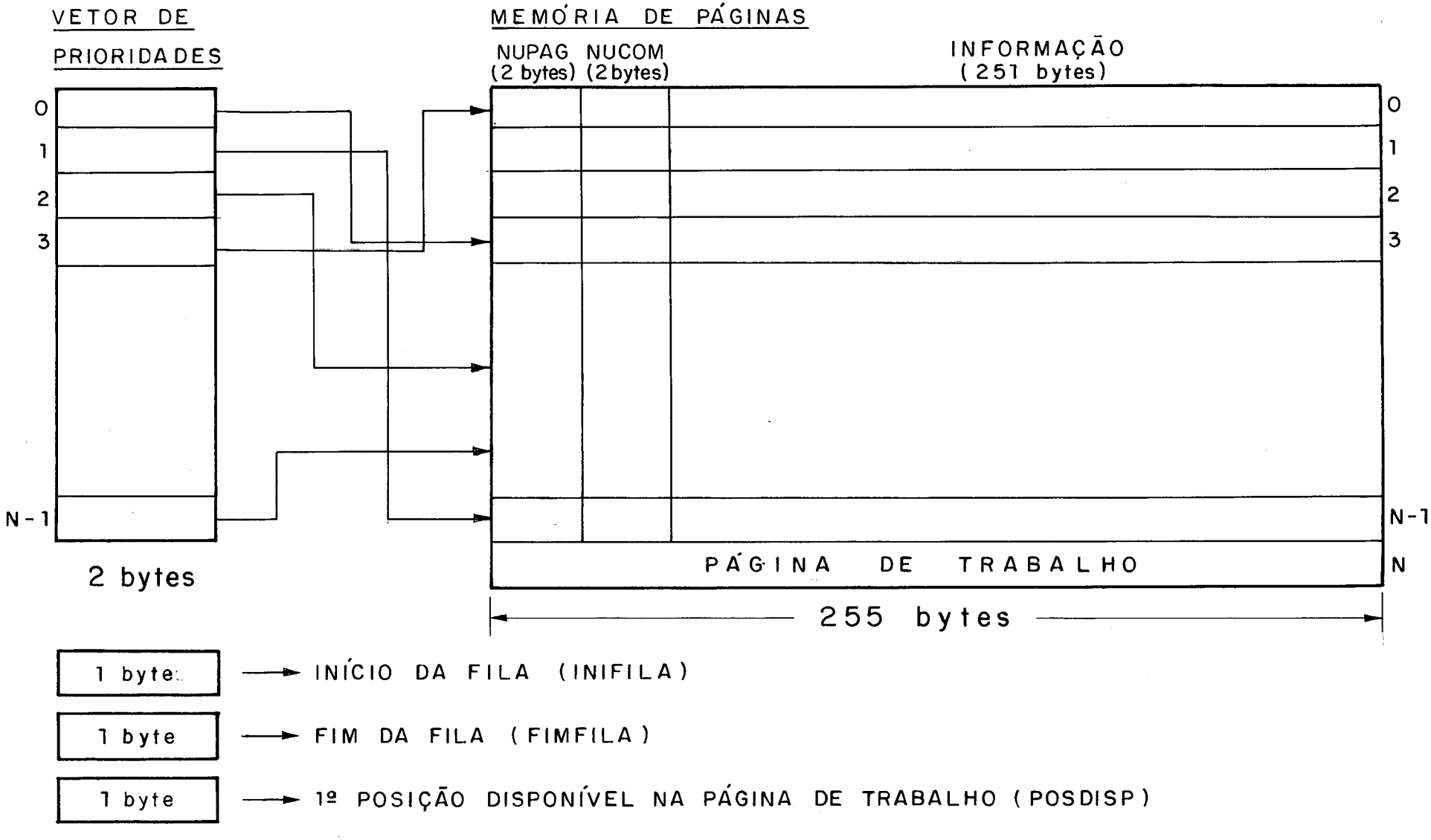


Figura II.8 - Esquematização da Área de Paginação.

b) Vetor de Prioridade (VETPRI)

É responsável por guardar informações de controle, necessárias ao funcionamento da paginação.

Cada uma das entradas (2 bytes) do vetor refere-se a uma das páginas guardadas na memória e contêm as seguintes informações:

. byte 1 - contêm informações gerais sobre a página correspondente. Dependendo do valor, são as seguintes essas informações:

0 - Informa que a página não foi alterada e que pode sair da memória através do "*page-out*";

1 - Informa que a página não foi alterada e que não pode sair da memória através do "*page-out*";

2 - Informa que a página foi alterada e que pode sair da memória através do "*page-out*"; e

3 - Informa que a página foi alterada e que não pode sair da memória através do "*page-out*".

. byte 2 - aponta para a entrada na memória de páginas onde se encontra a página correspondente.

De maneira a controlar também a prioridade de permanência na memória de cada uma das páginas aí guardadas, o Vetor de Prioridades funciona como uma "fila", onde cada página trazida para a memória entra na extremidade "fim da fila" (máxima prioridade de permanência) e cada página devolvida ao armazenamento secundário pelo "*page-out*", sai da extremidade "início da fila" (mínima prioridade

de permanência).

De modo a funcionar como uma fila, como falado acima, o Vetor de Prioridades será tratado como uma "lista linear circular", com os bytes "INICIO DA FILA" e "FIM DA FILA" mostrados na figura II.8, informando o início e o fim da fila, respectivamente. O funcionamento de tal tipo de estrutura de dado encontra-se muito bem descrito em |Knuth[23]| e |Santos[11]|

- MÉTODO DE PAGINAÇÃO

É através da memória de páginas que o sistema tem acesso aos dados do armazenamento secundário. Assim, sempre que precisar de uma página física, o sistema deverá ativar as primitivas de paginação (seção II.7), as quais se encarregarão de trazê-la para a memória e de devolver a entrada onde a mesma foi carregada.

De maneira a diminuir o número de acessos ao armazenamento secundário, o sistema tenta manter na memória as páginas mais frequentemente referenciadas, removendo rapidamente as poucas referenciadas.

Quando é feita uma referência a uma página física, o algoritmo de paginação é chamado e faz uma busca na memória de páginas de maneira a verificar se a mesma já se encontra aí. Dependendo do resultado desta busca, um dos seguintes procedimentos é executado:

a) A página não se encontra na memória.

Neste caso, o algoritmo verifica se a "fila" está cheia. Se não, procura uma entrada vazia na memória de páginas e inclui um ponteiro para esta na extremidade "fim da fila" (prioridade máxima). A passagem de a para b na figura II.9, ilustra este caso.

Se a fila já está cheia, a página de menor prioridade (apontada pela entrada "INÍCIO DA FILA") deverá ser removida da memória de páginas, de modo a liberar uma entrada para a página desejada. Esse processo de remoção implicará na gravação da página removida no registro correspondente no arquivo SOM, se esta foi alterada (valor 2 para o byte 1 da entrada correspondente em VETPRI). De maneira a atribuir a prioridadade máxima de permanência à nova página carregada, os ponteiros INÍCIO e FIM avançam (circularmente) uma entrada de forma a representar a nova situação da fila. A passagem de b para c na figura II.9 ilustra este caso.

Este procedimento de sempre dar prioridade máxima de permanência à página recém-carregada é uma aplicação da política de "mover para frente" em "arquivos auto-organizáveis" |Kunuth[23]|

b) A página referenciada já se encontra na memória.

Nesse caso, o algoritmo simplesmente devolve a entrada na memória de páginas onde a página desejada se encontra e de maneira a registrar a referência feita, aumenta de 1 a sua prioridade de permanência. Esse aumento de prioridade é conseguido trocando-se a entrada que aponta para a página re

ferenciada, pela imediatamente maior em prioridade. A passagem de c para d (troca dos valores das entradas 5 e 6 de VETPRI) na figura II.9, ilustra este caso. Este tipo de procedimento é conhecido como política de "transpor de 1" |Knuth[23]|.

Apesar dos procedimentos apresentados acima serem executados automaticamente pelo sistema, duas maneiras de se fugir deles são possibilitadas, de forma a que se possa influir diretamente na política de paginação com base em informações mais precisas, e não simplesmente na aleatoriedade de referências. Assim, as rotinas semânticas e primitivas podem influir explicitamente na política de paginação através dos seguintes procedimentos:

a) Bloqueio de uma página à remoção.

De modo a garantir que uma página não sairá da memória, esta pode ser bloqueada à remoção, através da atribuição do valor 1 ou 3 para o byte 1 da entrada correspondente em VETPRI, devendo a mesma ser liberada quando não for mais indispensável. Assim, se no "*page-out*" a página a remover da memória (menor prioridade) está bloqueada, o sistema anda na fila (no início → fim) até encontrar a primeira não bloqueada, liberando a entrada que esta ocupa para a página a ser carregada. O sistema se encarrega de remanejar as prioridades. Este procedimento se encontra ilustrado na passagem de a para b na figura II.10.

O sistema deve controlar muito bem o bloqueio e liberação de

páginas, de forma a evitar que chegue um momento em que todas as páginas estão bloqueadas ("*deadlock*").

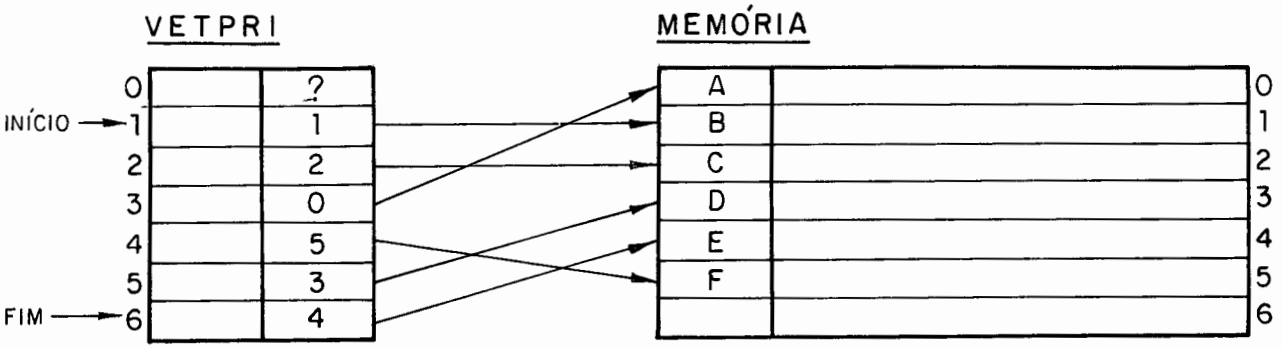
b) Especificação de prioridade para a página referenciada.

Se a prioridade da página referenciada é especificada, esta prevalece sobre as políticas de "mover para frente" e "transpor de 1" como apresentadas, e a página ficará exatamente com a prioridade especificada (0 → N-1), independente de se ela já se encontrava ou não na memória. Em qualquer das situações as prioridades são automaticamente rearranjadas, de modo a refletir a nova situação.

O procedimento que trata tal tipo de referência encontra-se programado nas primitivas de paginação (ver Anexo), e é ilustrado pela passagem de b para c na figura II.10.

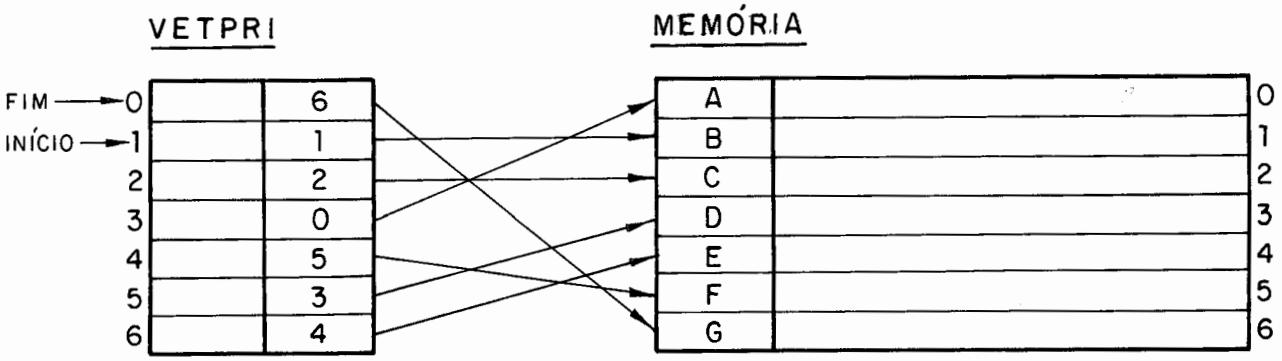
Uma outra característica importante a ser notada na área de paginação mostrada na figura II.8 é a presença de uma "página de trabalho" logo após as reservadas para a paginação, e de um byte chamado de "1ª POSIÇÃO DISPONÍVEL NA PAG. de TRABALHO (POSDISP)". A função desta entrada extra na memória de páginas, é servir como área de trabalho onde informações derivadas nas rotinas do SGBD a serem inseridas na Base de Dados Armazenada (ex.: tuplas internas, número de pág. física, ect)serão montadas e daí copiadas para as demais entradas da memória de páginas. Com isso, o fluxo de informações sempre será da memória de páginas para ela mesma, ainda que sejam dados a serem incluidos na Base de Dados Armazenada pelas rotinas semânticas ou primitivas. O byte POSDISP tem a função de informar a cada momento onde começa a área disponível da página de trabalho, uma vez que esta é usada simultâneamente por várias rotinas.

a)



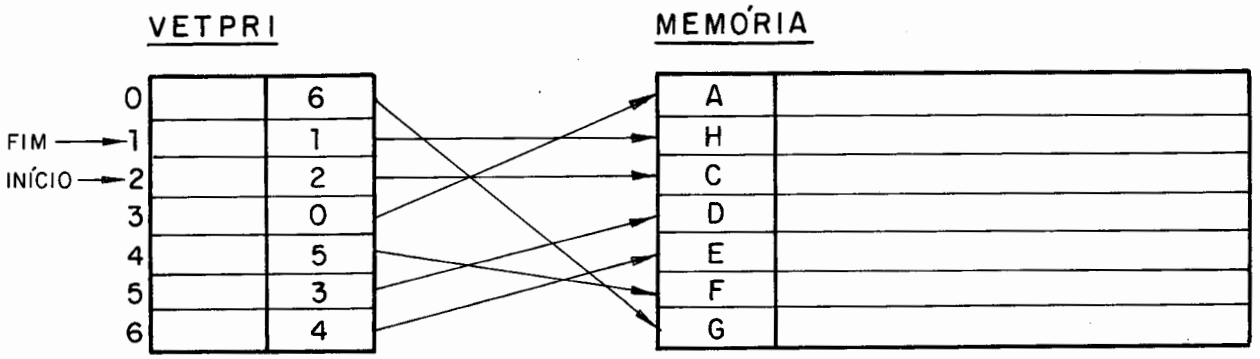
PRIORIDADE CRESCENTE - B, C, A, F, D, E

b) Incluir G



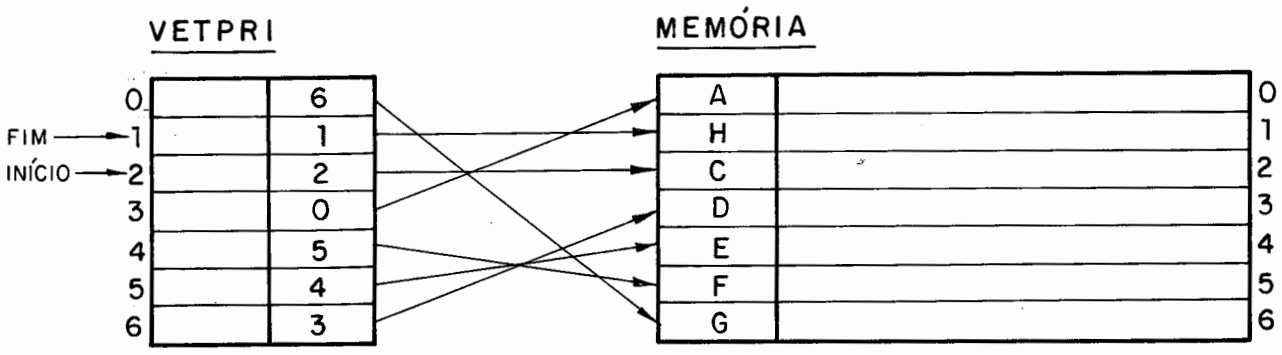
PRIORIDADE CRESCENTE - B, C, A, F, D, E, G

c) Incluir H



PRIORIDADE CRESCENTE - C, A, F, D, E, G, H

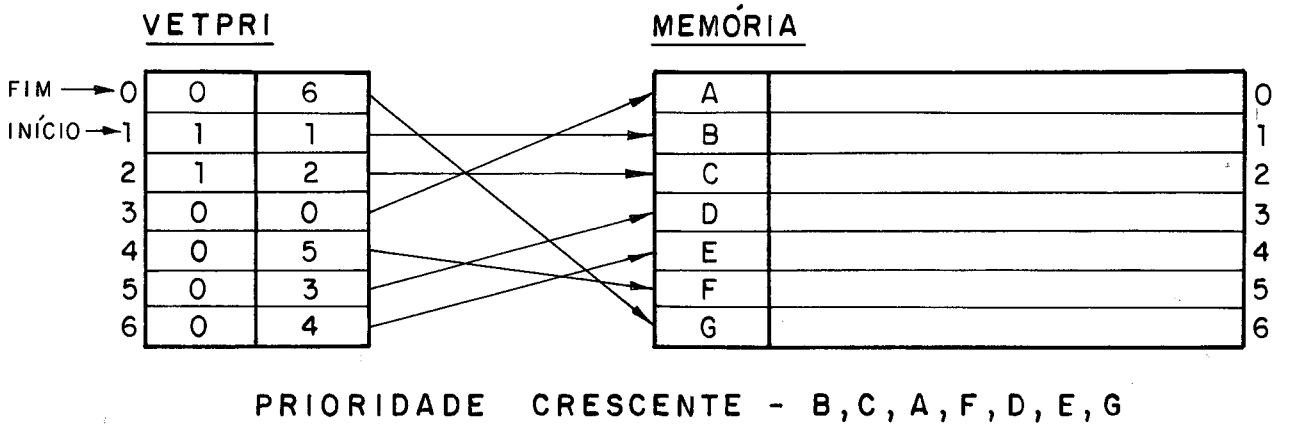
d) Referencia a D



PRIORIDADE CRESCENTE - C, A, F, E, D, G, H

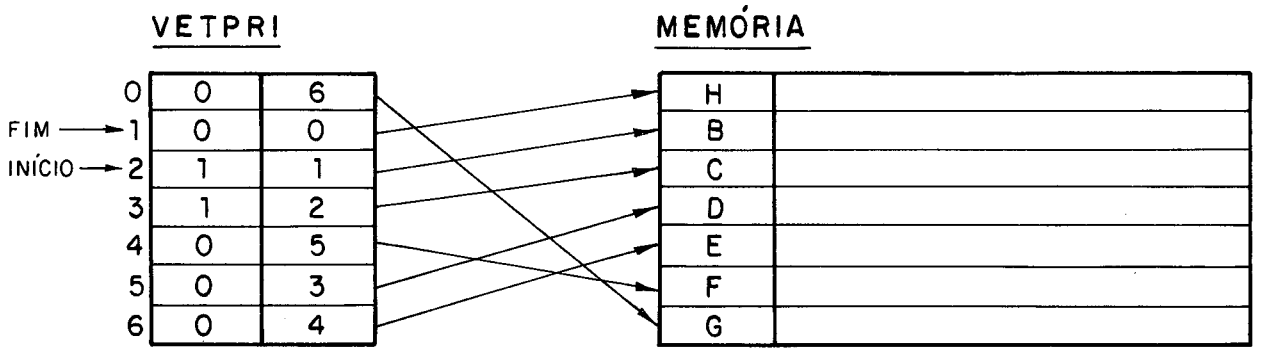
Figura II.9 - Exemplo 1 de funcionamento do método de paginação (para N=7).

a) Estado Inicial (p/ o Exemplo)



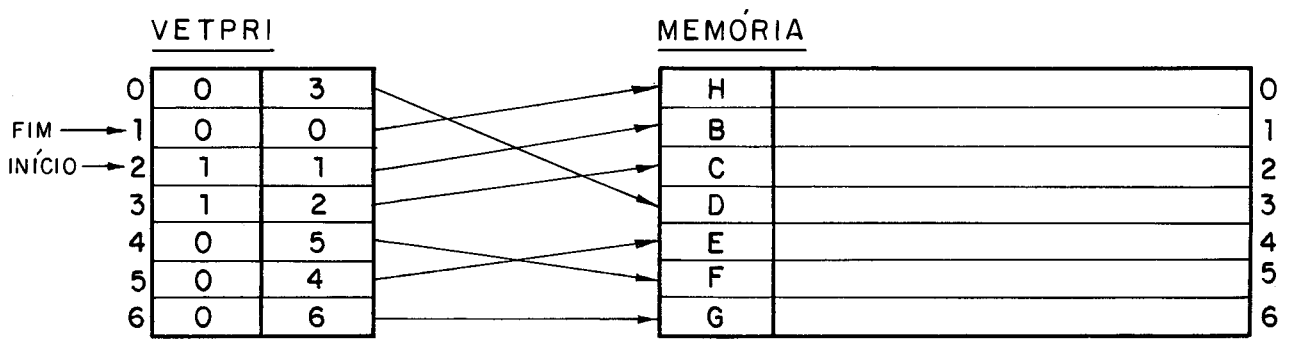
PRIORIDADE CRESCENTE - B, C, A, F, D, E, G

b) Incluir H



PRIORIDADE CRESCENTE - B, C, F, D, E, G, H

c) Referencia o D com prioridade 5



PRIORIDADE CRESCENTE - B, C, F, E, G, D, H

Figura II.10 - Exemplo 2 de funcionamento do método de paginação (para N = 7).

II.6. RECONSTRUÇÃO DA BASE DE DADOS ARMAZENADA

Um dos recursos mais importantes normalmente disponíveis em Sistemas de Banco de Dados, é a possibilidade de restauração dos dados, até algum estado atrás no tempo.

A provisão de um bom método de "Reconstrução da BDA" é muito útil tanto na garantia da integridade dos dados presentes à Base de Dados, quanto como uma ferramenta geral que possibilite a volta dos dados a um estado anterior no tempo, a partir de um estado não desejado.

De maneira a permitir a reconstrução da BDA, um histórico das alterações feitas sobre esta deverá ser mantido, o qual informará os diversos estados passados da BDA, para os quais se pode querer que ela seja reconstruida. Esse histórico normalmente referido por "*fita-log*" na literatura, também algumas vezes será chamado de "Cadeia de Reconstrução" que é o termo MICROLOBAN para ele.

Da mesma forma que a BDA, a Cadeia de Reconstrução também deverá ser constituida de páginas de tamanho fixo, e o tamanho deverá ser igual ao das páginas da BDA, de forma a receber as cópias antigas destas quando as mesmas são alteradas.

Assim na nossa implementação MICROLOBAN, a Cadeia de Reconstrução também será constituida de páginas de 255 bytes, e pelas mesmas razões da BDA, também será armazenada através de um arquivo relativo SOM e ficará restrita a uma unidade de disquete.

- FUNCIONAMENTO DA RECONSTRUÇÃO

Quando uma página da BDA é alterada, a cópia antiga com o número do comando que a alterou preenchido no campo NUCOM, é copiado na Cadeia de Reconstrução, logo após o último registro gravado, ficando a atualizada na própria Base de Dados Armazenada. Com isto consegue-se não apenas o histórico da BDA, mas também uma ordenação entre as páginas registradas de acordo com o número do comando que as alterou.

A gravação do número do comando que alterou a página junto com a cópia antiga desta é muito importante uma vez que são os comandos quem demarcam os vários estados passados da BDA. Isso significa que ela pode ser reconstruida até o estado que se encontrava após a execução de um determinado comando passado no tempo, na sessão atual. Este tipo de reconstrução é chamado de "Reconstrução de Comandos" em contraste à "Reconstrução de Sessões", a qual será explicada na seção II.6.2.

II.6.1. RECONSTRUÇÃO DE COMANDOS

Duas são as formas de se pedir a reconstrução da BDA com base nos comandos da sessão atual. A "Reconstrução Progressiva" especifica que a partir de um determinado estado passado da BDA (início da sessão atual na nossa implementação), apenas o efeito dos I primeiros comandos deverá ser mantido ; enquanto que a "Reconstrução Regressiva" especifica que a partir do estado atual da BDA, o efeito dos J últimos comandos

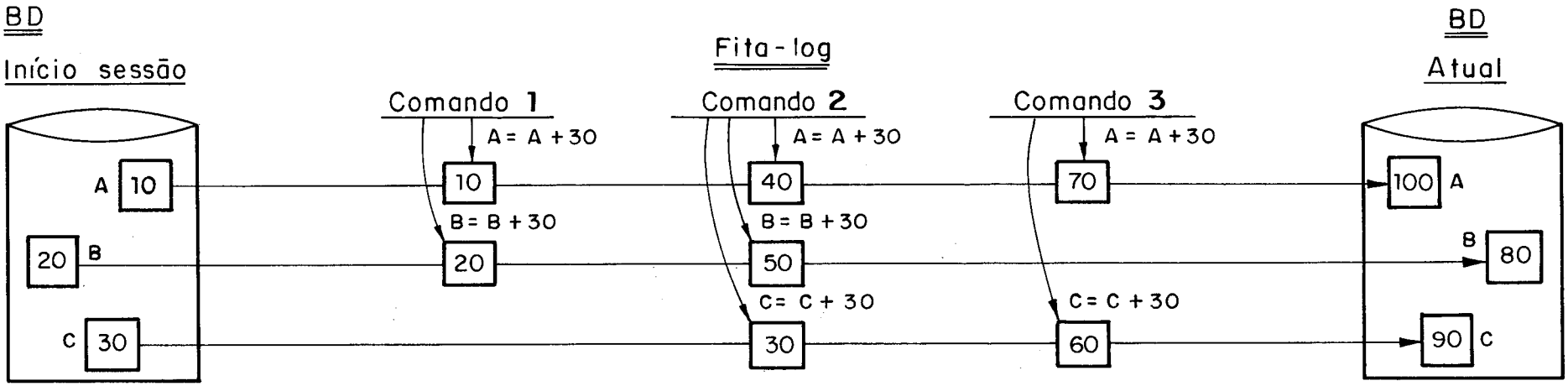
deverá ser desfeito.

Apesar das duas formas de especificação de reconstrução, a verdade é que um procedimento único pode ser assumido, ou seja, a BDA deverá voltar para o estado em que ela se encontrava após a execução de um comando L fixado, independente de se ter pedido a Reconstrução Progressiva ou Regressiva.

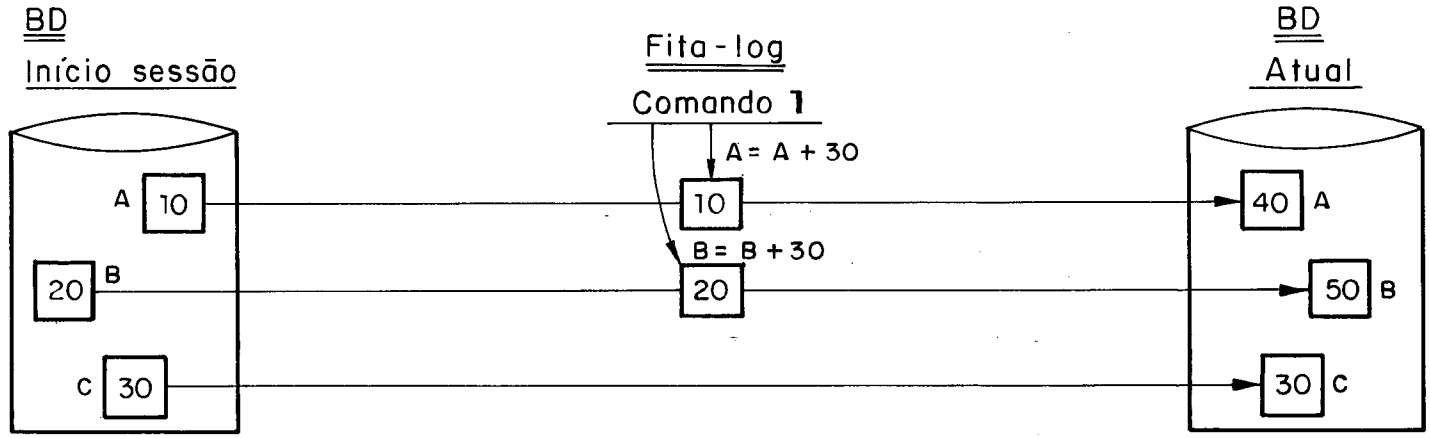
Em virtude de sempre mantermos as cópias atualizadas das páginas na BDA e as antigas na Cadeia de Reconstrução, o procedimento a ser executado na reconstrução sempre implicará na substituição das cópias atuais (na BDA) pelas antigas, o que caracteriza a Reconstrução Regressiva, ou seja, a partir do estado atual chegar ao estado em que a BDA se encontrava após a execução de um comando L fixado.

A figura II.11 dá um exemplo da Reconstrução de Comandos proposta.

a) Antes da Reconstrução



b) Reconstruir até depois do comando 1



Obs: **1** - Apenas as cópias iniciais 30, 40, 50 de cada página na "fita-log", a partir do comando especificado, é que foram restauradas. As cópias intermediárias 60, 70 foram simplesmente desprezadas;

2 - A cadeia de reconstrução também é atualizada, através da remoção de todas as páginas alteradas após o comando especificado.

Figura II.11 - Exemplo da Recontrução de Comandos.

II.6.2. RECONSTRUÇÃO DE SESSÕES

A possibilidade de reconstruir a Base de Dados Armazenada até um estado anterior na mesma sessão como apresentado, é um recurso muito importante tanto para o usuário que está trabalhando interativamente, como para o próprio SGBD na garantia da integridade dos dados; porém nenhuma utilidade tem para o Administrador da Base de Dados (ABD) na sua tarefa de controle geral sobre o uso da Base de Dados. Assim de maneira a também funcionar como uma ferramenta a serviço do ABD, a Reconstrução de Sessões, ou seja, a possibilidade de reconstruir a BDA até o estado em que esta se encontrava entre duas sessões passadas no tempo, é proposta de maneira a que o ABD possa desfazer o efeito inconveniente de uma ou várias sessões inteiras sobre a Base de Dados.

Uma primeira observação importante a ser feita sobre a Reconstrução de Sessões é quanto ao próprio conceito de sessão. Aqui nos referimos à "Sessão da Base de Dados" como o tempo decorrido entre a abertura e o fechamento da Base de Dados para uso, e não à sessão em um terminal. Assim sempre que uma Base de Dados é aberta, o seu "número de sessão atual" é incrementado de 1, de forma a diferenciá-la da anterior.

Em virtude das páginas da Cadeia de Reconstrução registrarem apenas os comandos que as alteraram sem conter nenhuma informação sobre a sessão, informações a respeito das sessões registradas deverão ser mantidas de modo a se promover a Reconstrução de Sessões. A figura II.12 esquematiza as informações de controle necessárias não apenas ao funcionamento da re

construção de sessões, mas à reconstrução de um modo geral.

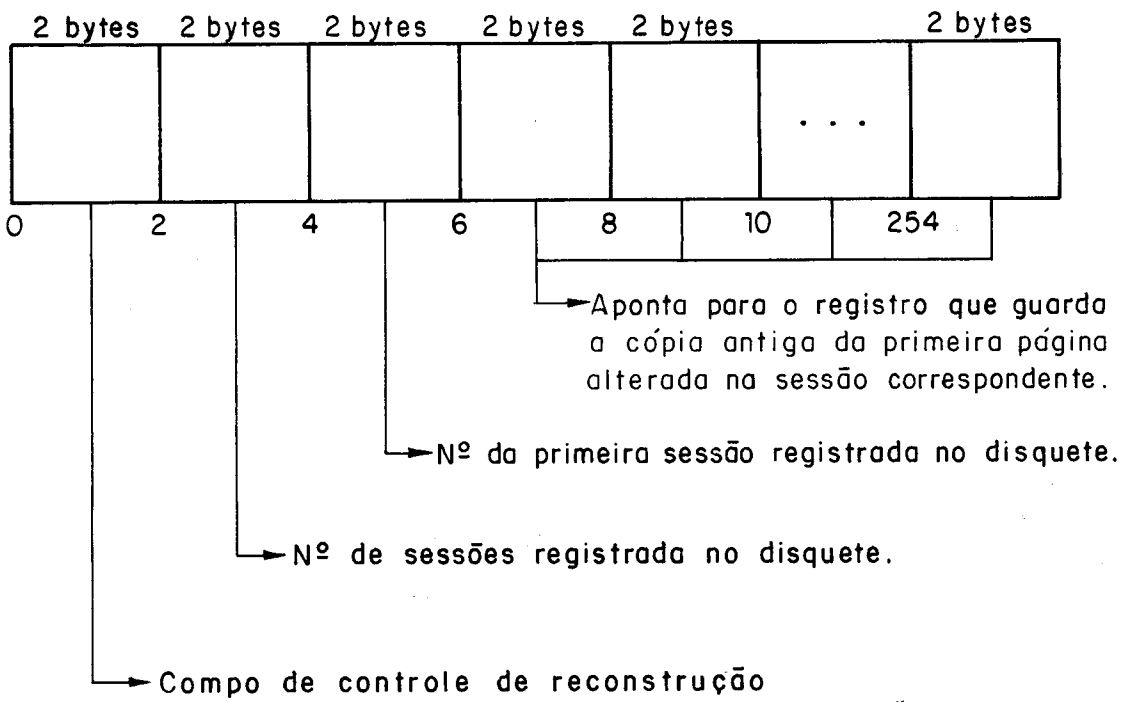
O Diretório de Reconstrução mostrado na figura II.12 é quem se encarrega de indicar quais as sessões que estão registradas no disquete e o registro relativo onde começam as alterações de cada uma delas no disquete de reconstrução. Com essas informações, quando pedida uma reconstrução da BDA até o estado em que ela se encontrava após uma determinada sessão, o SGBD descobre onde começam as alterações feitas na sessão seguinte à especificada e a partir daí procede como na Reconstrução de Comandos.

Uma observação importante a ser feita é que apenas a cópia inicial de cada página alterada na sessão é que tem utilidade para a Reconstrução de Sessões. Assim de forma a ganhar espaço na fita-log, quando a sessão é fechada, todas as cópias intermediárias das páginas alteradas na sessão são removidas.

a) Diretório de Reconstrução (DIRREC)

Ocupa o primeiro registro relativo no disquete de Reconstrução.

Formato:



b) Campo de Controle de Reconstrução

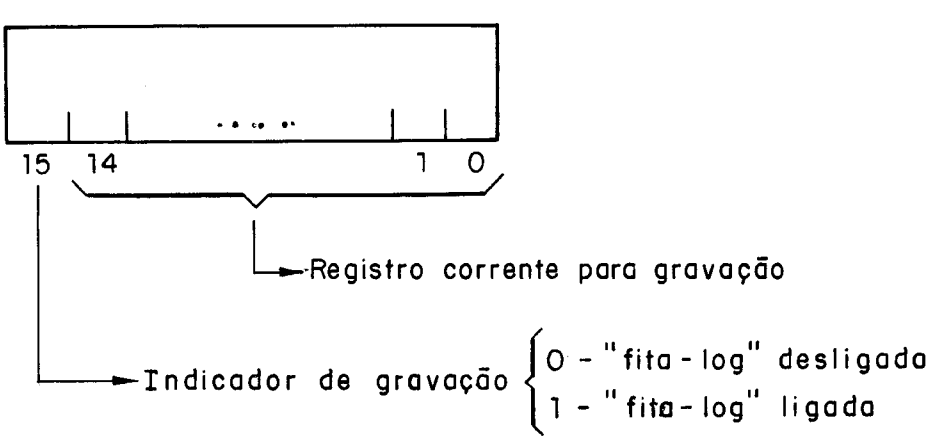


Figura II.12 - Informações de Controle para a Reconstrução.

II.7. PRIMITIVAS DE GERENCIAMENTO INTERNO

De maneira a fechar a proposta de um Ambiente Interno a ser suportado na implementação de um SGBD, falta apenas definir as primitivas a serem chamadas pelas rotinas semânticas(nível conceitual) de forma a acessarem e manipularem as estruturas de dados a nível interno, as quais na arquitetura apresentada no capítulo I foram chamadas de *primitivas de Gerenciamento Interno*.

Nesta seção nós descreveremos as primitivas contidas no ambiente proposto, e o Anexo traz os algoritmos que programaram a adaptação destas à implementação MICROLOBAN.

Três são as classes de primitivas disponíveis, dependendo da parte do Ambiente Interno a que elas se destinam:

- primitivas de paginação;
- primitivas de acesso e manipulação de tabelas básicas;
- primitivas de reconstrução.

II.7.1. PRIMITIVAS DE PAGINAÇÃO

Se encarregam de realizar a gerência da área da memória reservada para receber as páginas trazidas do armazenamento secundário.

II.7.1.1. OBTENTPAG (INDCRI, NUPAGFIS, PRIORI, ENTMEM)

Objetivo - Devolver a entrada (0 a N-1) na Memória de Páginas onde se encontra uma página física (de número NUPAGFIS) desejada, trazendo-a do armazenamento secundário se a mesma não se encontra na memória ou criando-a se é uma página nova (INDCRI=1)

II.7.1.2. PAGE-OUT (ENTMEM)

Objetivo - Remover da memória de páginas a página com menor prioridade de permanência na memória, de acordo com o Vetor de Prioridades, gravando-a no armazenamento secundário se a mesma foi alterada.

II.7.1.3. MUDAPRI (ENT1, ENT2, ENTMEM)

Objetivo - Rearranjar as prioridades no Vetor de Prioridades de forma a registrar uma referência feita a uma das páginas que já se encontrava na Memória de Páginas, ou a entrada aí de uma página lida do armazenamento secundário.

II.7.1.4. COPIAR (ENTMEM, DESLOC1, ENTMEM2, DESLOC2, TAMANHO)

Objetivo - Copiar uma informação de TAMANHO bytes a partir da posição DESLOC1 da entrada ENTMEM1 da memória de Páginas para a entrada ENTMEM2, também da memória de Páginas, a partir da posição DESLOC2.

II.7.2. PRIMITIVAS DE ACESSO E MANIPULAÇÃO DE TABELAS BÁSICAS

Se encarregam do mapeamento entre o nível conceitual e o interno, no que chamadas pelas rotinas semânticas atuam diretamente sobre as estruturas de dados como internamente representadas, ou seja, sobre as tabelas básicas.

De maneira a atuar sobre as tabelas básicas, as primitivas necessitam das informações sobre essas, tais como tamanho da tupla, cardinalidade, etc, as quais são encontradas no Diretório de Tabelas Básicas. Assim de maneira a atuar, as primitivas normalmente receberão como parâmetros as entradas no Diretório de Tabelas Básicas que referem-se às tabelas desejadas.

II.7.2.1. ALOCPAGFIS (ENTMEM1, DESLOC1, NUPAGLOG, ENTMEM)

Objetivo - Alocar uma página física a partir da Pilha de Disponíveis para a tabela básica descrita pela tupla do Diretório de Tabelas Básicas que se encontra na entrada ENTMEM1 da Memória de Páginas, com deslocamento DESLOC1.

II.7.2.2. DESALOCPAG (ENTMEM)

Objetivo - Devolver para a Pilha de Disponíveis a página física que se encontra na entrada ENTMEM da Memória de Páginas.

II.7.2.3. LERPAGLOG (PRIMENT, NUPAGLOG, NUPAGFIS)

Objetivo - Devolver o número da página física correspondente à página lógica NUPAGLOG para a tabela básica que começa na entrada PRIMENT do Índice de Páginas.

II.7.2.4. LETUPPOS (POSREL, TAMANHO, PRIMENT, ENTMEM, DESLOC)

Objetivo - Ler a tupla de TAMANHO bytes e posição relativa POSREL da tabela básica formada a partir da entrada PRIMENT do Índice de Páginas.

II.7.2.5. LETUPTID (TID, PRIMENT, ENTMEM, DESLOC)

Objetivo - Ler a tupla cujo "tid"("*tuple identifier*") é TID da tabela básica formada a partir da entrada PRIMENT do Índice de Páginas.

II.7.2.6. LETUPCHAV (ENTMEM1, DESLOC1, VALOR, ENTMEM2, DESLOC2)

Objetivo - ler a tupla cujo valor apresentado para a chave primária é VALOR, da tabela básica descrita pela tupla do Diretório de Tabelas Básicas que está na posição (ENTMEM1, DESLOC1) da Memória de Páginas.

II.7.2.7. INCTUP (ENTMEM1, DESLOC1, ENTMEM2, DESLOC2, RESULT)

Objetivo - Incluir a tupla que se encontra a partir da posição (ENTMEM2, DESLOC2) da Memória de Páginas, na tabela básica descrita pela tupla que se encontra na posição (ENTMEM1, DESLOC1).

II.7.2.8. REMTUP (ENTMEM1, DESLOC1, ENTMEM2, DESLOC2, NUPAGLOG)

Objetivo - Remover a tupla que se encontra a partir da posição (ENTMEM2, DESLOC2) da Memória de Páginas, da tabela básica descrita pela tupla que está em (ENTMEN1, DESLOC1).

II.7.2.9. CRIATAB (ENTMEM, DESLOC)

Objetivo - Criar uma tabela básica de acordo com as informações registradas na tupla que está a partir da posição (ENTMEM, DESLOC) da Memória de Páginas.

II.7.2.10. REMOVETAB (ENTMEM, DESLOC)

Objetivo - Remover a tabela básica que é descrita pela tupla que está a partir da posição (ENTMEM, DESLOC) da Memória de Páginas.

II.7.3. PRIMITIVAS DE RECONSTRUÇÃO

Se encarregam de realizar os procedimentos que suportarão o esquema de reconstrução proposto.

II.7.3.1. RECSESSÃO (NUSEC)

Objetivo - Reconstruir a Base de Dados Armazenada até o estado que esta se encontrava após o término da sessão NUSEC do ACSET correspondente.

II.7.3.2. RECCOMANDO (NUCOMANDO)

Objetivo - Reconstruir a Base de Dados Armazenada até o estado em que esta se encontrava após a execução do comando NUCOMANDO nesta mesma sessão.

II.7.3.3. LIMPAFITA

Objetivo - Eliminar da fita-log as cópias intermediárias das páginas, relativas à sessão corrente.

CAPÍTULO III

III. CONSTRUÇÕES SUPORTADAS PELA INTERFACE MICROLOBAN

Neste capítulo nós tentamos dar uma visão geral das estruturas de dados suportadas pela interface MICROLOBAN, as quais, semelhantemente à LOBAN, são referidas como construções.

As construções se compõem de outras construções e assim sucessivamente, até chegar ao nível mais elementar, ou seja, ao nível denominado atômico.

De maneira a descrever as construções suportadas por MICROLOBAN, nós partiremos das construções mais simples e através do processo de composição chegaremos as construções mais abrangentes possíveis.

Chamamos a atenção para o fato que a especificação da interface não é o objetivo desta tese, e portanto não abordaremos os conceitos básicos com grande profundidade, mas apenas estabelecemos uma base para os futuros capítulos. Detalhes suplementares poderão ser encontrados em |Santos[11,12]|.

III.1. PRETIPOS ATÔMICOS MANTIDOS

Da mesma forma que em LOBAN, um "pretipo" é um tipo padrão predefinido como parte integrante das convenções a nível de interface, através dos quais os usuários formam novos tipos de dados de acordo com suas necessidades.

Os pretipos atômicos (indivisíveis) mantidos em MICROLOBAN são:

a) INTEIRO - é o conjunto matemático dos números inteiros. É representado por uma cadeia de dígitos com ou sem sinal à esquerda.

b) REAL - é o conjunto matemático dos números reais. É representado por uma cadeia de dígitos com uma vírgula, com ou sem sinal.

c) SIGLA - é representado por uma cadeia de caracteres de qualquer tamanho (1 a 80) informado. Diferentemente de LOBAN, as siglas MICROLOBAN necessitam de informações a respeito da sua representação (número de caracteres).

d) DATA - considerando a importância geral da informação indicando o dia de um mês de um ano, foi previsto o pretipo (com designação padrão) DATA. Diferentemente de LOBAN, MICROLOBAN considera uma construção do pretipo DATA como sendo atômica, e por isso a informação por ela representado só pode ser acessado como um todo. A representação para uma construção do pretipo DATA é DDMMAA onde DD, MM e AA são números inteiros e representam valores válidos para dia, mês e ano respectivamente.

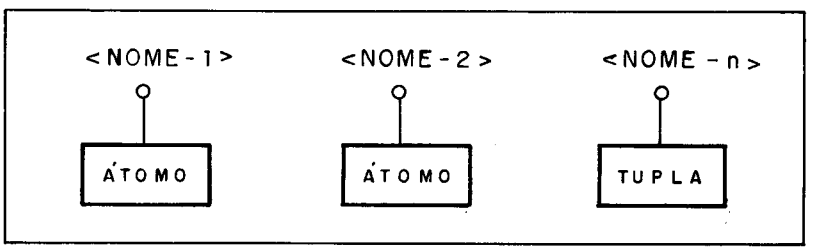
e) HORA - representa combinações válidas de valores válidos para hora, minuto e segundo, e como DATA também será considerado atô-

mico por MICROLOBAN. Uma construção do pretipo DATA é representada por HHMMSS onde HH, MM e SS são números inteiros, porém só podem ser referenciados em conjunto.

III.2. TUPLAS

Uma tupla é uma nominação (componentes imediatos têm nomes) sobre átomos ou tuplas. É importante observar que MICROLOBAN permite apenas tuplas com dois níveis de embutimento, ou seja, se uma tupla é componente imediato de outra, seus componentes imediatos têm que ser átomos.

A figura III.1 ilustra uma composição possível de uma tupla.



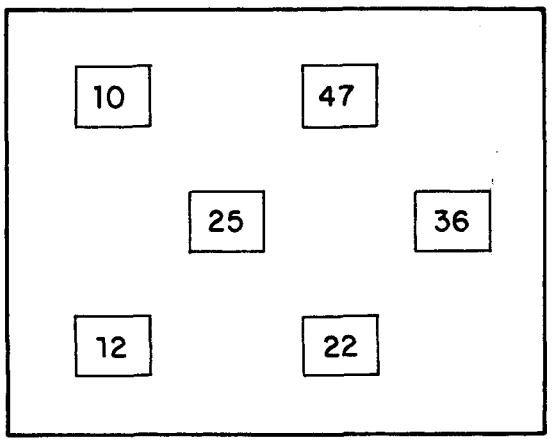
Obs.: A tupla sob nome <nome-m> só pode ser composta de átomos de forma a manter o limite no número de embutimentos.

Figura III.1 - Ilustração de uma tupla.

III.3. ITENS E COLEÇÕES DE ITENS

O pretipo item tendo designação padrão ITEM, em MICROLOBAN é formado pelo conjunto de construções definido pelos pretipos atômicos: inteiro (INTEIRO/INT), real (REAL), sigla (SIGLA), data (DATA) e hora (HORA).

O pretipo com designação padrão COLITENS é definido por uma coleção (componentes imediatos sem nome) cujos componentes imediatos sejam construções do pretipo item, pertencentes ao mesmo tipo. A figura III.2 ilustra um exemplo de uma coleção de itens.



Obs.: Os componentes imediatos da coleção de itens mostrados são do pretipo inteiro.
Todos os itens devem ser do mesmo tipo, como por exemplo inteiros entre 0 e 50.

Figura III.2 - Exemplo de uma coleção de itens (COLITENS).

III.4. LIGAÇÕES E TABELAS

III.4.1. TABELAS

Diz-se que uma tabela é uma coleção no que seus componentes imediatos são construções sem nome.

Todavia uma tabela não é uma coleção qualquer, pois além de seus componentes imediatos, tuplas ou ligações, serem nominações, os nomes são comuns. Assim os nomes dos componentes imediatos de uma tabela se aplicam a todas as construções de uma mesma coluna, podendo então serem fatorados para referirem-se a todos os elementos de uma coluna como pode-se ver na figura III.3.

A		B	C	D			...
A1	A2			D1	D2	D3	

Figura III.3 - Representação gráfica de uma tabela.

Na figura III.3 os nomes dos componentes imediatos (A,B,C,D ...) dos componentes imediatos da própria tabela são mostrados, e ainda os componentes imediatos (A1,A2,D1,D2,D3) a esses que também são nominações (A,D), constituindo assim uma hierarquia.

A uma sequência de nomes obedecendo a uma destas hierarquias, que permitem associar uma coluna ou conjunto de colunas, chama-se atributo.

No exemplo da figura III.3 são atributos , por exemplo, tanto as sequências de nomes A.A1, D.D2 e D.D3, como as sequências A, B, C e D, mas não D1, D2, D3, A1 e A2.

Mais uma vez chamamos a atenção para a limitação MICROLOBAN de no máximo dois níveis de embutimento para tuplas, o que força que as construções sob nomes A1, A2, D1, D2 e D3 no exemplo da figura III.3 obrigatoriamente sejam átomos (indivisíveis).

III.4.2. TABELA RELACIONAL (TAREL)

Uma tabela relacional em MICROLOBAN é uma coleção homogênea de tuplas do mesmo tipo. A figura III.4 apresenta a representação gráfica de uma tabela relacional.

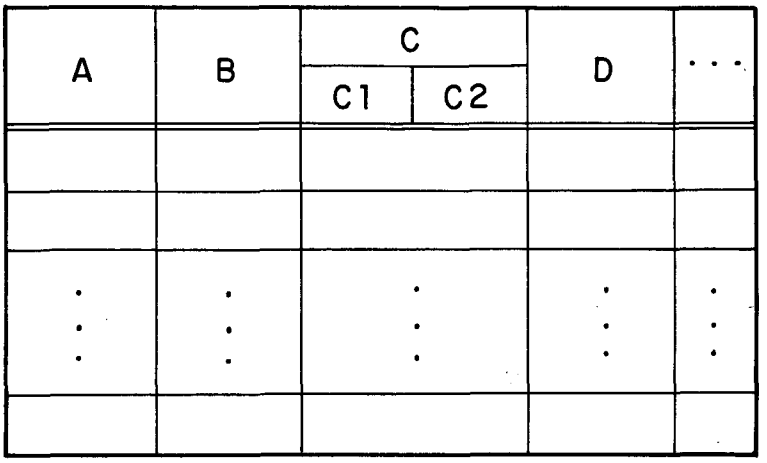


Figura III.4 - Representação gráfica de uma Tabela Relacional.

Tabela Relacional é o termo MICROLOBAN para referir as relações como apresentadas normalmente no modelo relacional, e cada linha da tabela é uma tupla na terminologia relacional, ou um registro em outras terminologias de processamento de dados.

Aplicam-se as tabelas relacionais todas as observações já feitas para tabelas em geral.

Uma tabela relacional caracteriza-se por apresentar um conjunto de um ou vários atributos, que é declarado como chave primária para esta no que conhecendo-se o seu valor, tem-se acesso direto à tupla correspondente. No máximo a chave compreende toda a tupla, uma vez que uma tabela relacional nunca possui duas tuplas exatamente iguais em valor. A informação à respeito das chaves para as tabelas relacionais deve ser fornecida na descrição dos tipos dessas tabelas, e é muito importante para a implementação, como falado no capítulo anterior.

III.4.3. LIGAÇÃO (LIG)

É uma nominação sobre uma tupla (sob nome L padrão) dita Ligante e uma tabela relacional (sob nome T padrão) como acima definida, dita Tabela Ligada.

A figura III.5 representa graficamente uma ligação.

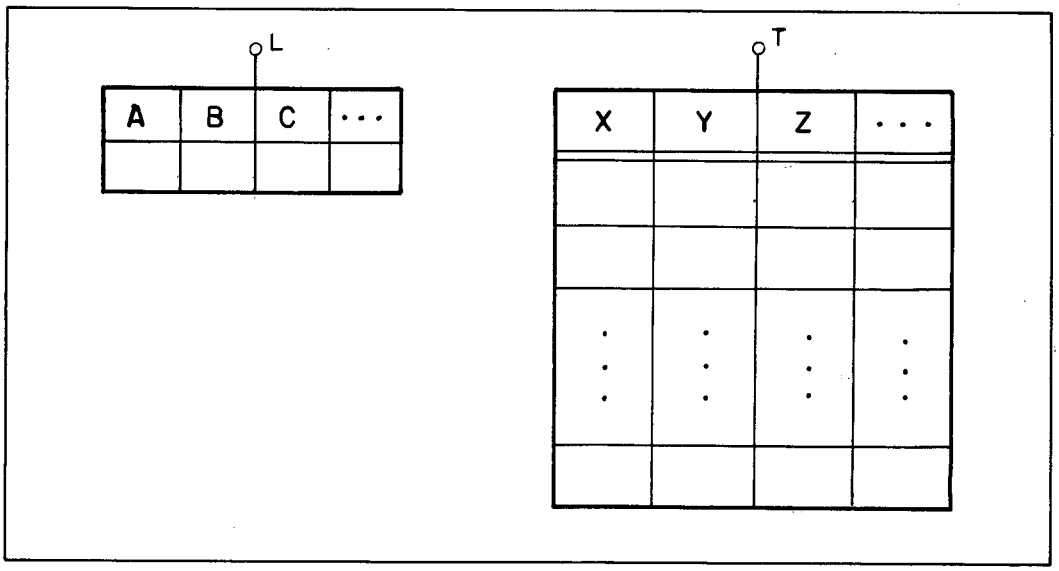


Figura III.5 - Representação gráfica de uma ligação.

III.4.4. TABELA LIGACIONAL (TALIG)

É uma coleção homogênea de Ligações do mesmo tipo. A figura III.6 representa graficamente uma tabela ligacional.

Entre outras observações, aplicam-se as tabelas ligacionais todas essas já feitas para as tabelas em geral.

Da mesma foram que uma tabela relacional, a ligacional também caracteriza-se por apresentar um conjunto de um ou vários atributos da tupla ligante, cujo valor identifica univocamente cada ligação da tabela ligacional, sendo portanto a sua chave primária. No máximo a chave primária para uma tabela ligacional será toda a tupla ligante, uma vez que não podem haver dois ligantes iguais em valor numa mesma tabela ligacional.

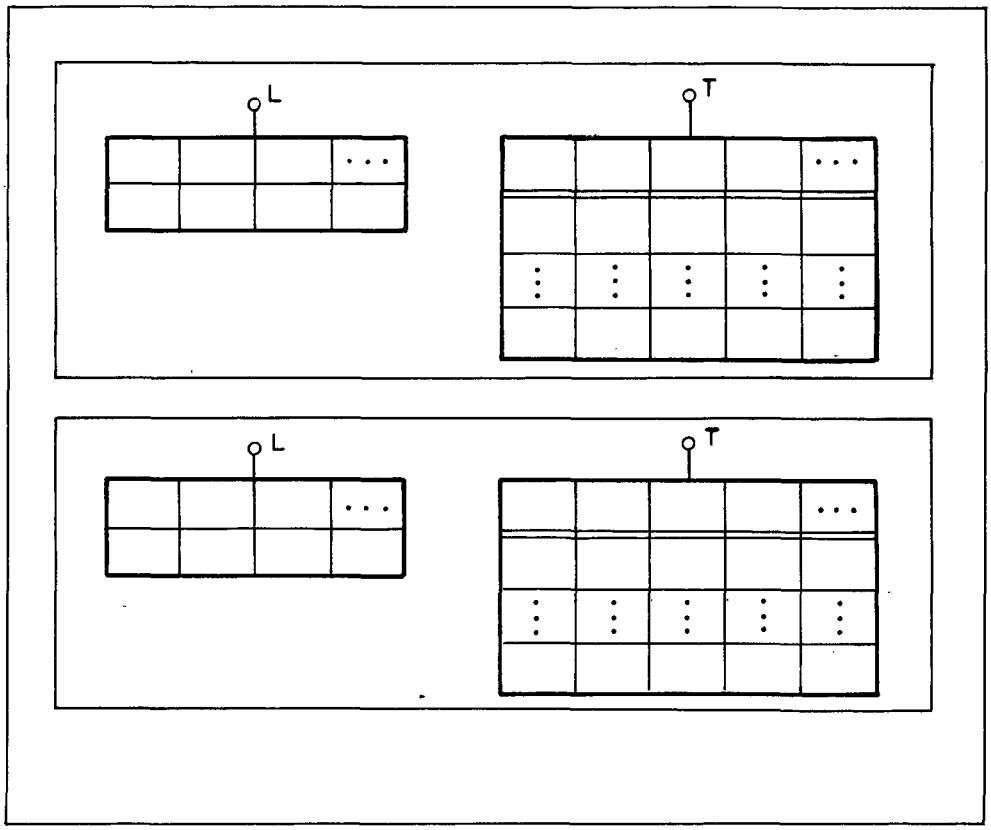


Figura III.6 - Representação gráfica de uma tabela ligacional.

III.5. COMPOSIÇÃO DA FOLHA (FOLHA)

Como LOBAN, MICROLOBAN caracteriza-se por apresentar as descrições dos dados presentes ao acervo (esquema conceitual segundo |Date[22]|) armazenadas como construções enderecáveis e manipuláveis pelos usuários, desde que estes tenham autorização para isso. Essas descrições encontram-se em forma de verbetes |Santos[11,12]|, e são armazenadas na FOLHA, juntamente com os verbetes de acesso que definem autorização sobre os dados presentes ao acervo, verbetes de usuários que identificam usuários para o sistema e verbetes de fonte que predefinem procedimentos através de texto-fonte.

A FOLHA é composta por quatro tabelas, uma para cada tipo de verbete, as quais têm nome e tipo padrão predefinido.

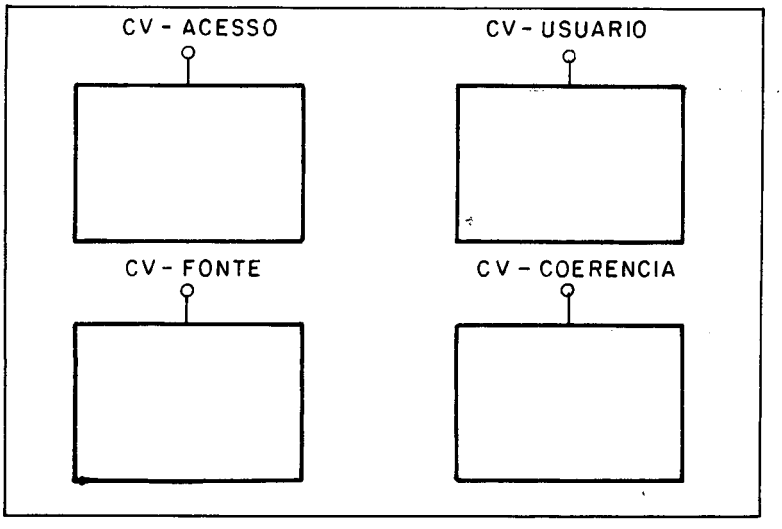
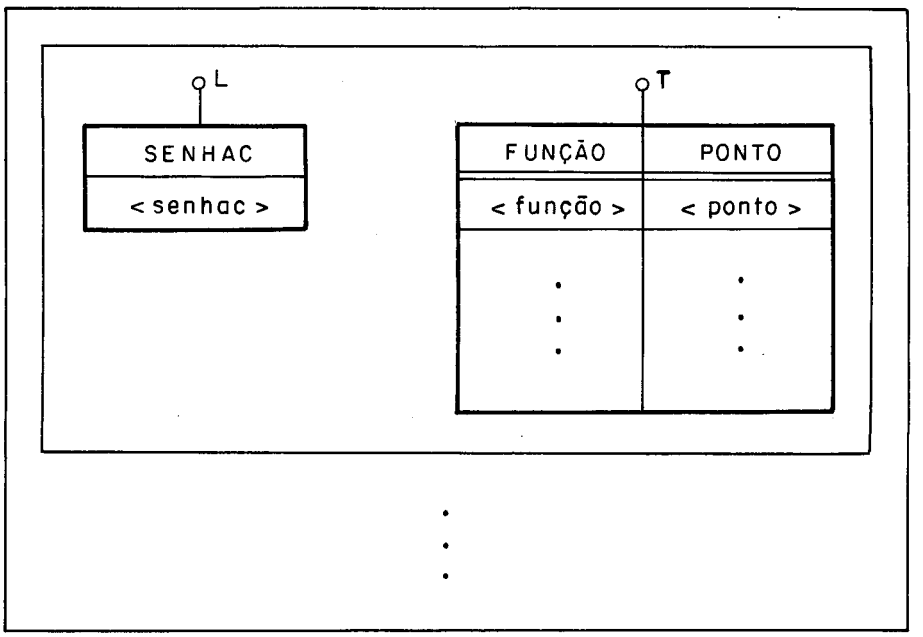


Figura III.7 - Representação gráfica da FOLHA.

III.5.1. TABELA DE ACESSO (CV-ACESSO)

É usada para armazenar os verbetes de acesso presentes à FOLHA. Cada verbete será armazenado através de uma ligação com o formato mostrado na figura III.8.



onde:

. <senhac> é o nome especificado para a senha de acesso. É chave primária para CV-ACESSO;

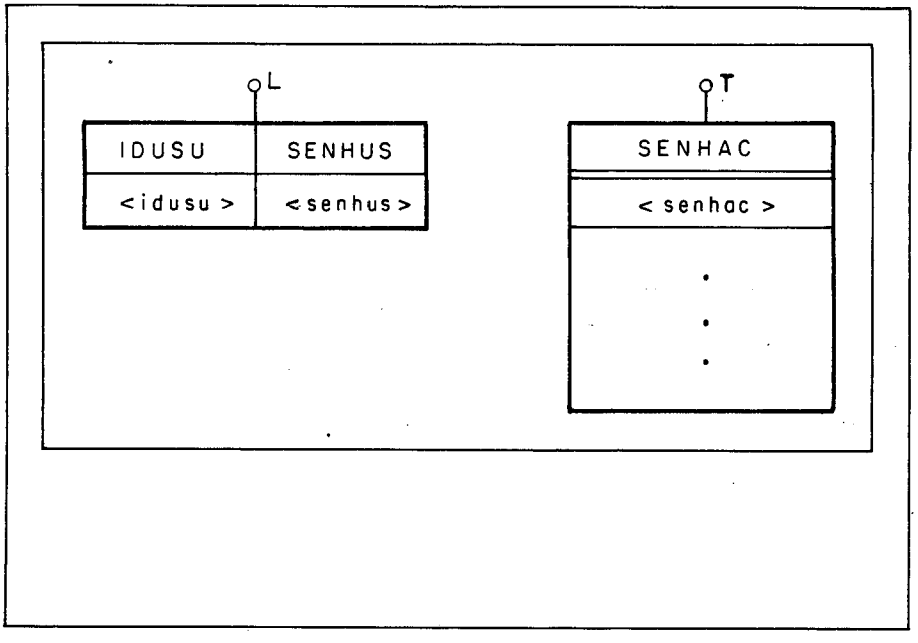
. <função> é o valor que informa qual a função de acesso (CONSULTAR, MODIFICAR, GERENCIAR) autorizada;

. <ponto> é o ponto sobre o qual se está autorizado. Pode receber os valores ACTRAB, FOLHA, FICHA ou o nome de qualquer um dos arquivos que compõem o ACTRAB, assim como qualquer uma das tabelas da FOLHA (CV-ACESSO, CV-USUÁRIO, CV-FONTE e CV-COERÊNCIA).

Figura III.8 - Representação gráfica da tabela de acesso (CV-ACESSO).

III.5.2. TABELA DE USUÁRIOS (CV-USUÁRIO)

É responsável pelo armazenamento dos verbetes de usuário associados ao sistema. Cada verbete de usuário será armazenado através de uma ligação, como mostrado na figura III.9.



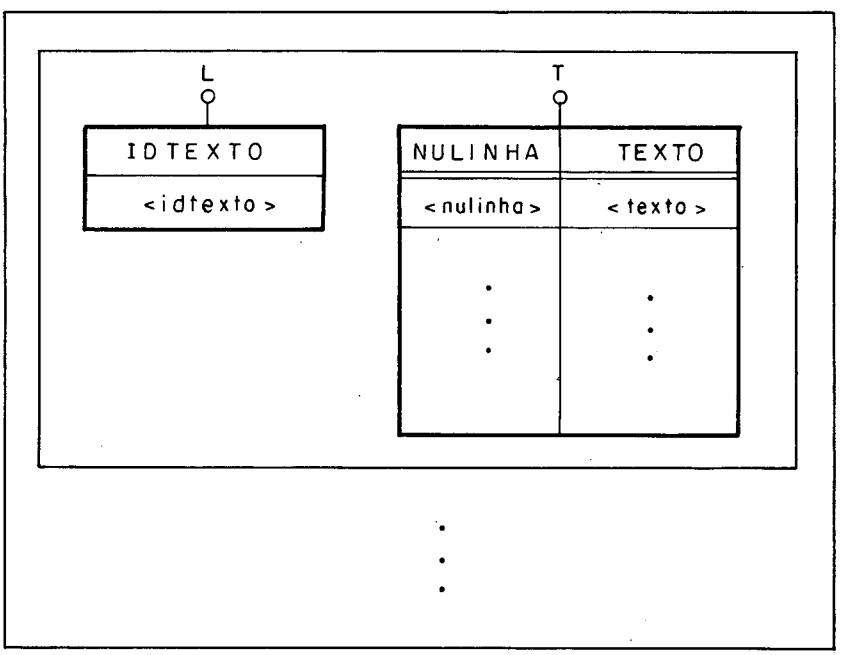
onde:

- <idusu> e <senhus> são a identificação e senha do usuário respectivamente. O atributo IDUSU é chave primária para a tabela CV-USUÁRIO.
- a tabela ligada apresenta um conjunto de senhas de acesso (autorização) associadas ao usuário, as quais devem existir em CV-ACESSO, ou seja, existe uma conexão |Santos[12]| entre CV-ACESSO e CV-USUÁRIO através do atributo SENHAC.

Figura III.9 - Representação gráfica da tabela de usuários (CV-USUÁRIO).

III.5.3. TABELA DE FONTE (CV-FONTE)

É usada para armazenar os verbetes de texto-fonte (procedimentos predefinidos) presentes ao acervo. Cada texto-fonte será armazenado através de uma ligação e diferentemente de LOBAN não apresenta parâmetros.



onde:

. <idtexto> é o nome dado ao texto fonte; e

. cada tupla ligada armazena uma linha do texto-fonte. <nulinha> é um átomo que á mantido pelo sistema e informa o número da linha no texto. Este átomo é aí inserido pelo sistema de modo a garantir que não haverão duas tuplas iguais, o que de outra maneira não estaria garantido. <texto> é cada uma das linhas que constituem o texto-fonte.

Figura III.10 - Representação gráfica para a tabela de fonte (CV-FONTE).

III.5.4. TABELA DE COERÊNCIA (CV-COERÊNCIA)

Armazena os verbetes de coerência na sua forma fonte, a qual pode ser acessada por usuários autorizados. Deve-se salientar contudo, que na implementação MICROLOBAN os verbetes de coerência também serão armazenados em uma forma intermediária mais adequada às rotinas de verificação da coerência. O armazenamento desta forma intermediária será tratado quando da apresentação do Esquema Interno no capítulo V.

A forma fonte de um verbete de coerência será armazenada através de uma ligação, como mostrado na figura III.11.

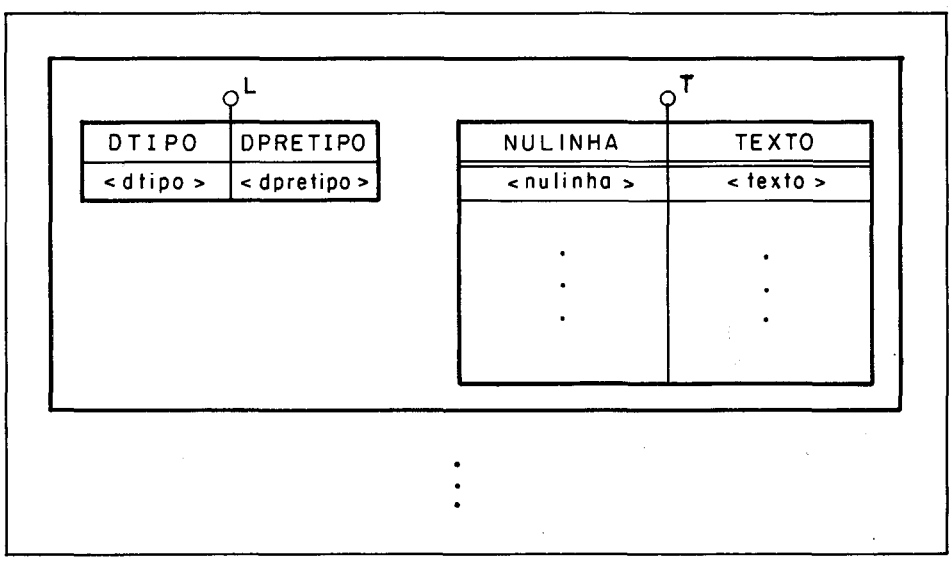


Figura III.11 - Representação gráfica da tabela de coerência (CV-COERÊNCIA)

Obs.: . <dtipo> é a designação do tipo informada no verbete. DTIPO é chave para CV-COERÊNCIA.

. <dpretipo> é a designação do pretipo no verbete;

. A tabela ligada funciona da mesma forma que em CV-FONTE, só que para verbetes de coerência.

III.6. ARQUIVOS

É uma nominação composta por uma construção sob nome FICHA e outra sob nome TR ou TL que pode ser uma tabela relacional ou ligacional.

III.6.1. COMPOSIÇÃO DA FICHA

A FICHA em MICROLOBAN é uma tupla de tipo e nome (FICHA) padrão (pretipo) que traz informações mantidas pelo sistema sobre os dados, informações essas que podem ser consultadas pelos usuários autorizados, porém não alteradas.

A FICHA, tanto de um arquivo quanto do ACTRAB (como será falada mais a frente neste capítulo), tem o mesmo tipo, o qual está mostrado na figura III.12.

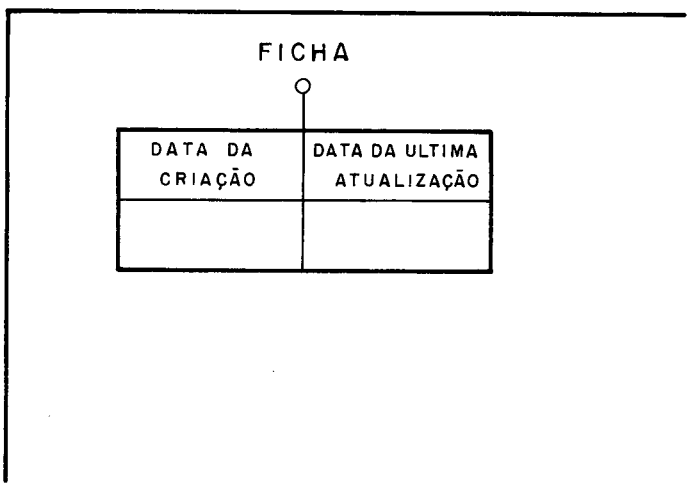


Figura III.12 - Representação gráfica da FICHA

Assim a FICHA de um arquivo armazena a "data de criação" e "data da última atualização" deste arquivo, informações essas preenchidas automaticamente pelo SGBD porém acessíveis aos usuários autorizados.

III.6.2. ARQUIVO RELACIONAL (AREL)

É aquele cuja tabela componente é relacional. A figura III.13 ilustra graficamente um arquivo relacional.

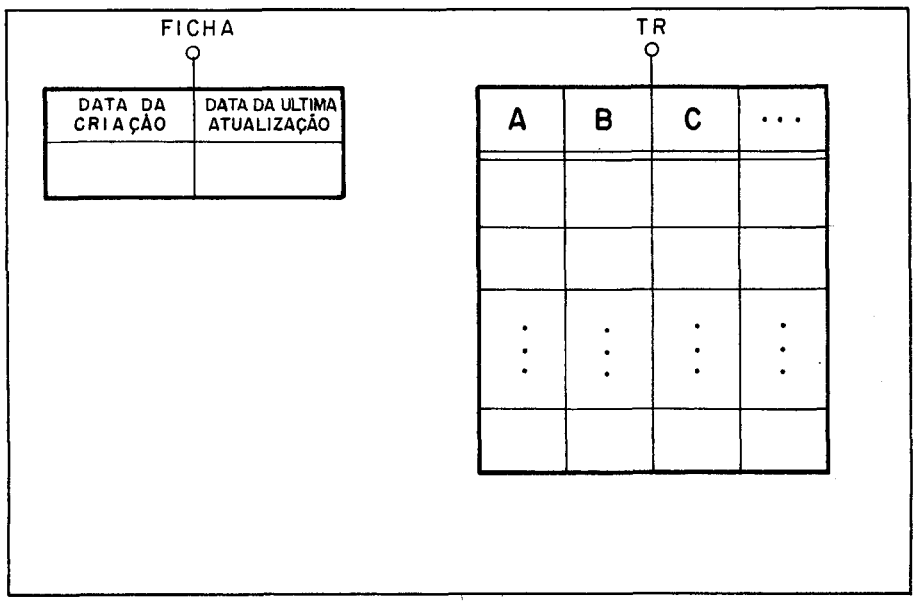


Figura III.13 - Representação gráfica de um Arquivo Relacional.

III.6.3. ARQUIVO LIGACIONAL (ALIG)

É aquele cuja tabela componente é ligacional.

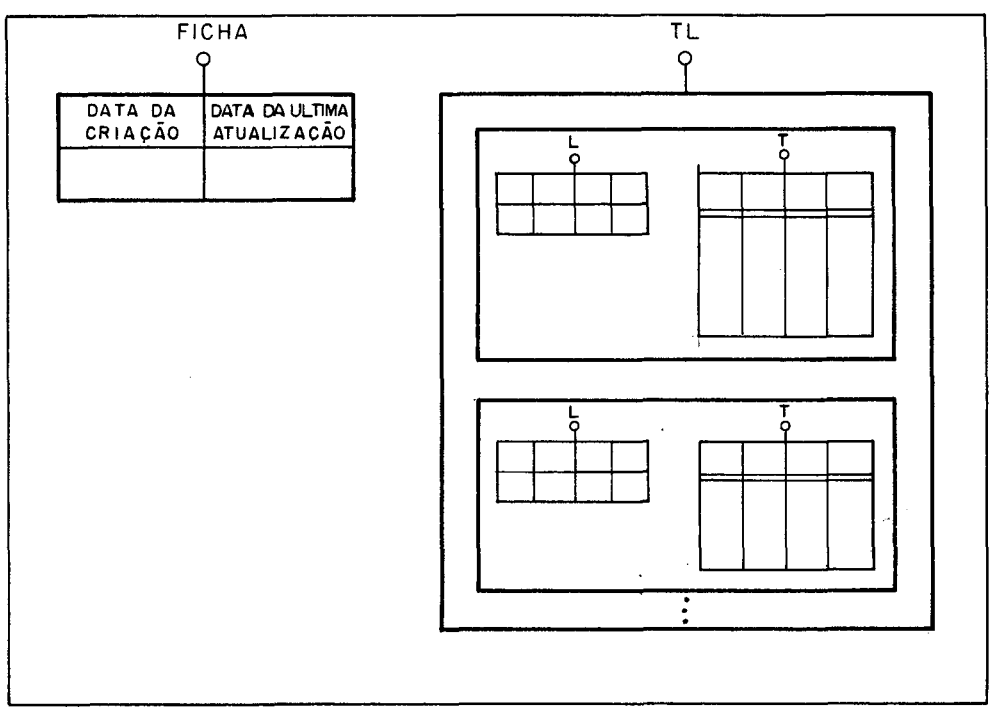


Figura III.14 - Representação gráfica de um Arquivo Ligacional.

Uma coisa importante a ser observada sobre os Arquivos MICROLOBAN é a ausência de relações de ordem como essas presentes em LOBAN.

III.7. COMPOSIÇÃO DO ACERVO DE TRABALHO (ACTRAB)

O Acervo de Trabalho (ACTRAB) é uma nominação cujos componentes imediatos são: uma construção sob nome e de pretipo FICHA como falado para arquivos, só que neste caso guardando informações sobre o ACTRAB; uma construção sob nome e pretipo FOLHA como apresentado na seção III.5 e nenhum, um, ou vários Arquivos Relacionais e/ou Ligacionais.

A figura III.15 esquematiza a composição do Acervo de Trabalho.

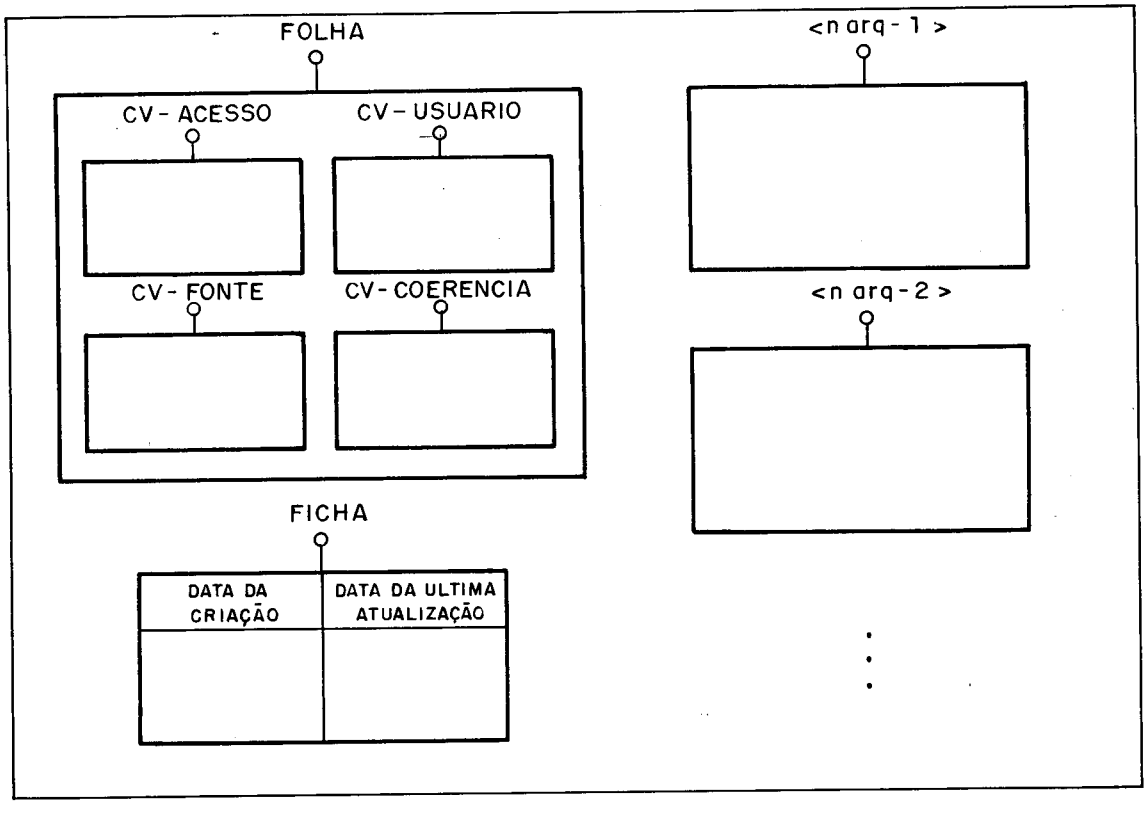


Figura III.15 - Representação gráfica para o Acervo de Trabalho.

Obs.: <n arq> representa o nome associado a cada um dos Arquivos, sejam eles relacionais ou ligacionais.

III.8. COMPOSIÇÃO DO ACERVO SETORIAL (ACSET)

Um Acerto Setorial é uma nominação cujos componentes imediatos são:

. O Acervo de Trabalho (ACTRAB); e

. Uma cadeia de reconstrução (CAD-REC) a qual informa as alterações feitas no ACTRAB de forma a que o mesmo possa ser reconstruido. A cadeia de reconstrução não é diretamente endereçável pelos usuários MICROLOBAN, mas apenas pelo SGBD tanto no registro das alterações quanto no procedimento de reconstrução do ACTRAB, como falado no capítulo passado.

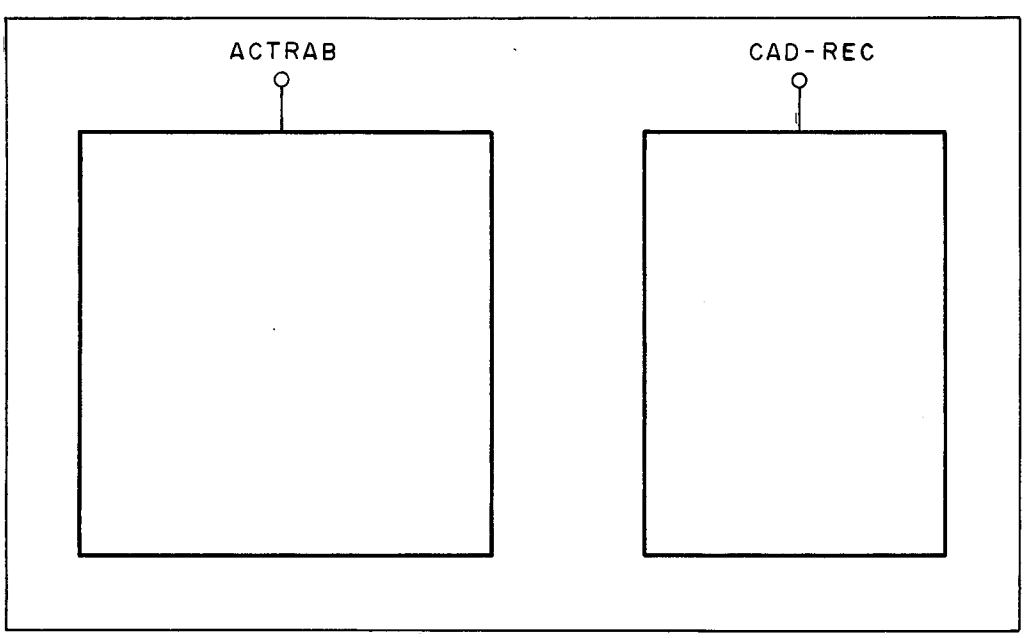


Figura III.16 - Representação gráfica do Acervo Setorial.

III.9. COMPOSIÇÃO DO ACERVO TOTAL

O Acervo Total MICROLOBAN é o que normalmente é referido como Base de Dados.

Dentro do Acervo Total estão os Acervos Setoriais como já definidos, os quais são conhecidos do sistema por nomes. Essa divisão em Acervos Setoriais facilita a implantação de aplicações, normalmente correspondendo um Acervo Setorial a uma aplicação ou sistema de informação.

Em MICROLOBAN além de não haver comunicação entre os Acervos Setoriais, apenas um destes pode estar aberto a cada momento, ainda que apenas para leitura.

A figura III.17 representa graficamente um Acervo Total, constituido pelos Acervos Setoriais FOLHA-PAGTO, CONSTR-ESTOQUE ..., VENDAS.

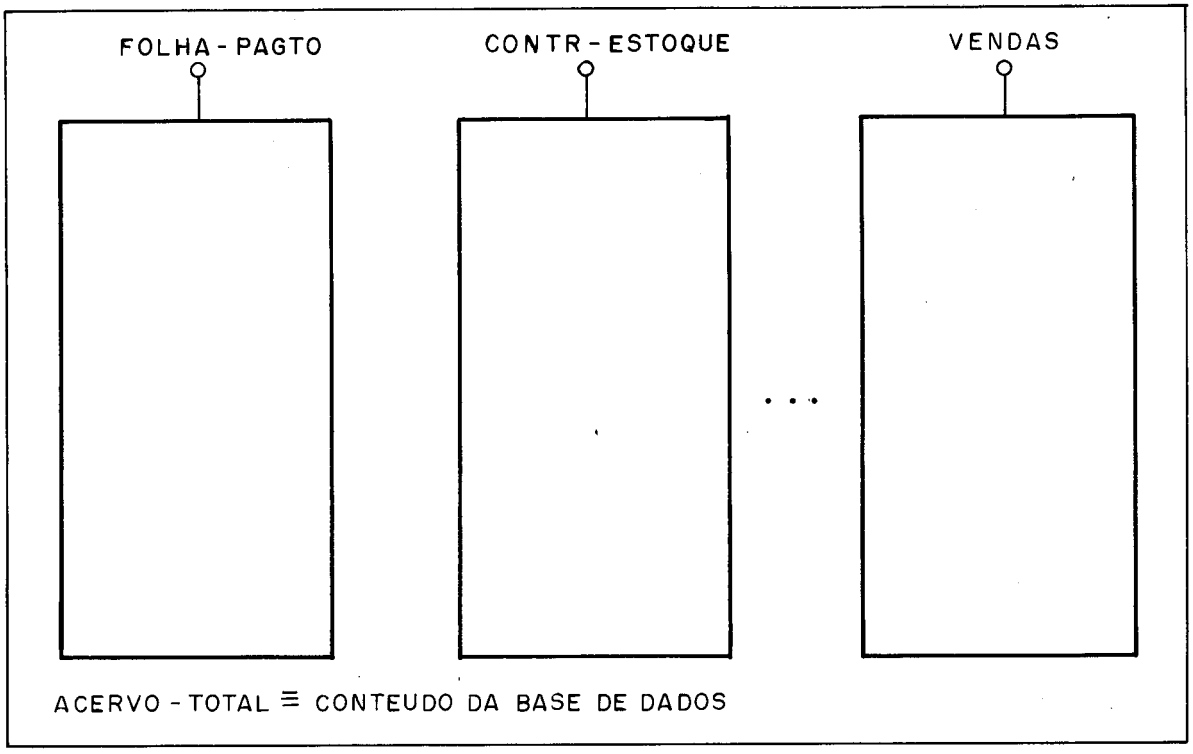


Figura III.17 - Representação gráfica do Acerto Total.

CAPÍTULO IV

IV. REPRESENTAÇÃO INTERNA DAS CONSTRUÇÕES MICROLOBAN

No capítulo III nós apresentamos as principais estruturas de dados (construções) e formas de navegação associadas, à disposição de um usuário MICROLOBAN, de forma a que este possa armazenar e recuperar os dados do seu interesse. No capítulo II apresentamos um Ambiente Interno geral, onde estão previstas estruturas básicas de armazenamento e métodos de acesso a serem suportadas no "nível interno" |ANSI|X3|SPARC[32]|, de forma a servirem como base de armazenamento e recuperação dos dados armazenados. O capítulo atual tem a função de realizar o mapeamento entre os dois capítulos acima citados no que ele define como as estruturas de dados conceituais (construções MICROLOBAN) são representadas através das internas (tabelas básicas) e como a navegação implícita às construções MICROLOBAN é internamente realizada.

IV.1. REPRESENTAÇÃO PARA TUPLAS

Sugere-se que as tuplas sejam armazenadas contiguamente e que a identificação de cada um dos atributos que a compõem seja feita através de um descritor dessa tupla. Dessa maneira as tuplas conceituais podem ser representadas através das tuplas básicas citadas no capítulo II, desde que um descritor adequado seja mantido de forma a possibilitar o acesso

a atributos individuais dessa tupla. A figura IV.1 representa graficamente o armazenamento de uma tupla.

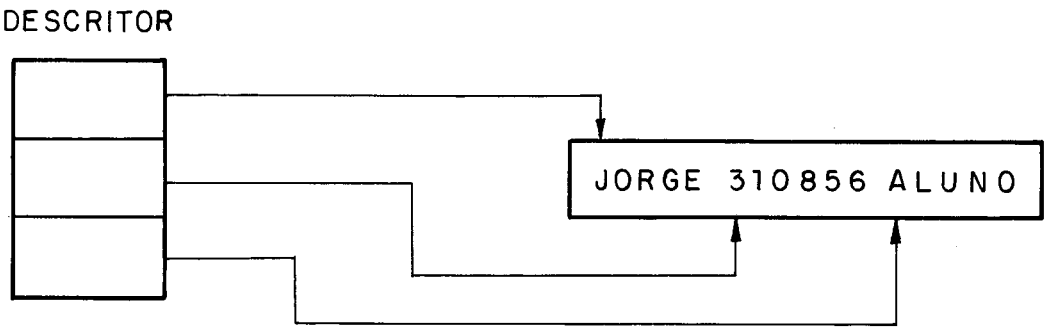


Figura IV.1 - Idéia gráfica do armazenamento de uma tupla.

O descritor mostrado acima foi colocado aí apenas como ilustração. A explicação detalhada não apenas de como são descritas as tuplas, mas todos os dados armazenados, será dada no próximo capítulo o qual trata especificamente dos "dados sobre dados".

IV.2. REPRESENTAÇÃO PARA TABELAS RELACIONAIS

Como deve-se ter percebido no capítulo III, "Tabela Relacional" é o termo adotado por MICROLOBAN para referir-se às Relações do modelo relacional. Assim como no capítulo II apresentou-se a tabela básica como sendo uma forma dirigida exatamente ao armazenamento das relações do modelo relacional, conclui-se imediatamente que as tabelas relacionais serão apresentadas internamente como tabelas básicas.

O processo de navegação MICROLOBAN inerente a uma ta

bela relacional é o sequencial, ou seja, é feita uma varredura (tupla a tupla) na tabela, até se encontrar a(s) tupla(s) desejada(s). Apesar de ser este também o método de acesso inerente a uma tabela básica, pode ser que uma otimização seja feita no sentido de diminuir o tempo de busca à tupla correspondente à tupla conceitual desejada, se alguma das condições abaixo é verificada:

a) Conhece-se o valor apresentado pelo atributo "chave primária" da tabela, na tupla desejada;

b) Conhece-se a "posição relativa" na tabela básica da tupla que armazena a conceitual desejada; ou

c) Conhece-se o TID ("tuple identifier") da tupla básica que armazena a tupla desejada.

Através do conhecimento de uma das características acima para a tupla desejada, o sistema se encarrega de acessar de forma "direta" (indexada na verdade, já que todo acesso a uma tabela básica tem que acessar primeiro o Índice de Páginas) a tupla básica que armazena esta, evitando todo o *"overhead"* implícito à varredura.

Uma coisa interessante a notar nos métodos de acesso acima listados, é que enquanto "posição relativa" e "TID" são características realmente internas, chave é conceitual. Assim o conceito de chave deve ser propagado às tabelas básicas, a partir das tabelas relacionais por estas representadas, de forma a possibilitar tal tipo de acesso "direto".

IV.3. REPRESENTAÇÃO PARA TABELAS LIGACIONAIS

Uma tabela ligacional foi definida no capítulo III como sendo uma coleção homogênea de ligações do mesmo tipo, onde cada uma destas é constituida de uma tupla (sob nome L) chamada ligante e uma tabela relacional chamada ligada (sob nome T).

De maneira a representar internamente uma tabela ligacional, três idéias principais são imediatamente formadas. De forma a ilustrar essas idéias, optar por uma delas e dar as justificativas desta escolha, tomaremos como base para representação a tabela ligacional apresentada na figura IV.2.

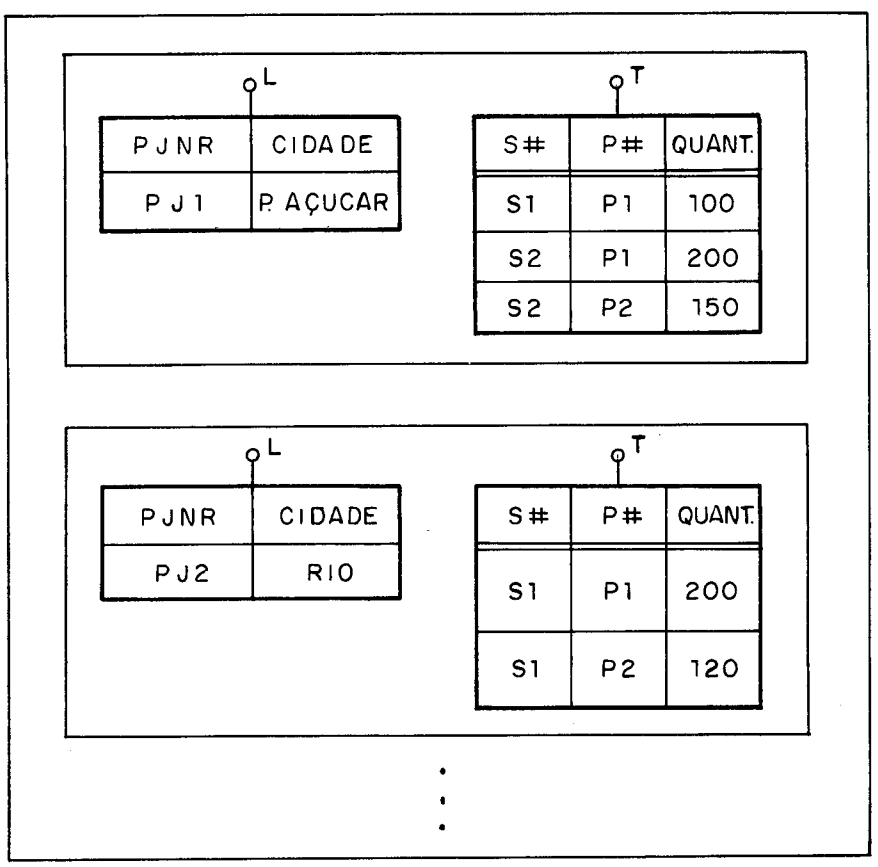


Figura IV.2 - Exemplo de Tabela Ligacional a ser representada.

IV.3.1. IDÉIA 1

Armazenar contiguamente na mesma área, os ligantes e ligados de uma tabela ligacional, sendo que cada ligante antecede fisicamente os "seus" ligados e aponta para o próximo ligante através de um ponteiro, de forma a possibilitar a varredura apenas dos ligante de uma tabela. Os ligados de um ligante devido à contiguidade física não precisariam de ponteiros. A figura IV.3 ilustra essa idéia.

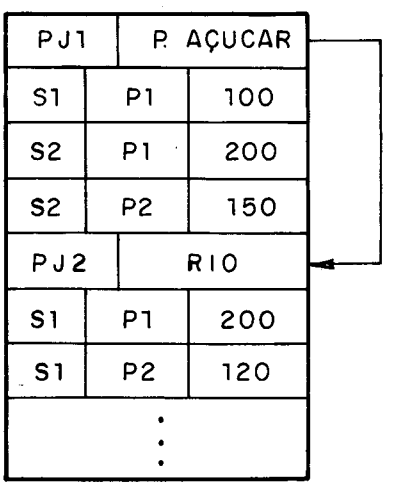


Figura IV.3 - Ilustração da idéia 1 para representação física de tabelas ligacionais.

Essa idéia tem vantagens e desvantagens bem evidentes, e nós as apresentamos agora de forma a justificar porque não a adotaremos.

a) Vantagens

. contiguidade das tuplas ligadas de uma mesma ligação.
. relativa economia de espaço devido ao pouco uso de ponteiros.
. apenas uma única organização para os dados.

b) Desvantagens

. devido à contiguidade, a inclusão e remoção implicará em deslocamentos à serem feitos nas tuplas.
. tuplas de diferentes tipos (ligante/ligados) a serem armazenadas pela mesma estrutura.

A primeira desvantagem poderia facilmente ser minimizada, através da dispensa da condição dos ligados estarem armazenados contiguamente (o que traria outras vantagens e desvantagens), porém nada faria com relação à segunda, a qual por si só já nos força a abandonar a idéia uma vez que o esquema proposto no capítulo II exige que as estruturas a serem armazenadas tenham apenas tuplas de um único tipo.

IV.3.2. IDÉIA 2

Em virtude do principal motivo pelo qual a idéia anterior não pode ser adotada, esta tenta propor um esquema parecido, só que tornando todas as tuplas do mesmo tipo, de modo a aproveitar o esquema de armazenamento/recuperação proposto no capítulo II. Assim a idéia aqui é transformar cada tabela liga

cional em relacional, de forma a se aproveitar da grande adequação destas ao ambiente proposto, como já falado.

A transformação de uma tabela ligacional em relacional é facilmente conseguida através da concatenação do ligante com cada um dos ligados, como pode ser visto na figura IV.4.

PJNR	CIDADE	S#	P#	QUANT
PJ1	P. AÇUCAR	S1	P1	100
PJ1	P. AÇUCAR	S2	P1	200
PJ1	P. AÇUCAR	S2	P2	150
PJ2	RIO	S1	P1	200
PJ2	RIO	S1	P2	120
⋮	⋮	⋮	⋮	⋮

Figura IV.4 - Tabela Relacional resultante da execução da Idéia 2.

A grande desvantagem inerente à esta idéia, é o espaço consumido pela repetição do ligante para cada um dos ligados a ele associado. Como o pré-requisito número 1 a ser seguido por implementações como esta que estamos descrevendo é a economia de espaço, tal idéia fica automaticamente eliminada, apesar das vantagens práticas associadas à sua adoção devido ao fato que as tabelas ligacionais seriam armazenadas exatamente como as relacionais.

IV.3.3. IDÉIA 3

Esta idéia, a qual foi a escolhida para ser posta em prática, tenta combinar as duas anteriores de uma forma que quase todas as vantagens delas sejam mantidas, porém minimizando as suas desvantagens e possibilitando a sua implementação através do esquema de armazenamento/recuperação do capítulo II.

Uma tabela ligacional será representada internamente como se fosse duas relacionais: uma para os ligantes e outra para os ligados. De maneira a manter o caminho de acesso do ligante para os "seus" ligados e de ligado para ligado de uma mesma ligação, alguns ponteiros serão acrescidos às tuplas de ligantes e ligados. Daí a tabela de ligantes apresentar o atributo LINK cuja finalidade é apontar para a entrada, na tabela dos ligados, onde se encontra o primeiro ligado desta ligação; e a tabela dos ligados apresentar RLINK e LLINK que se encarregam de encadear duplamente os ligados de uma mesma ligação. A figura IV.5 ilustra a idéia proposta.

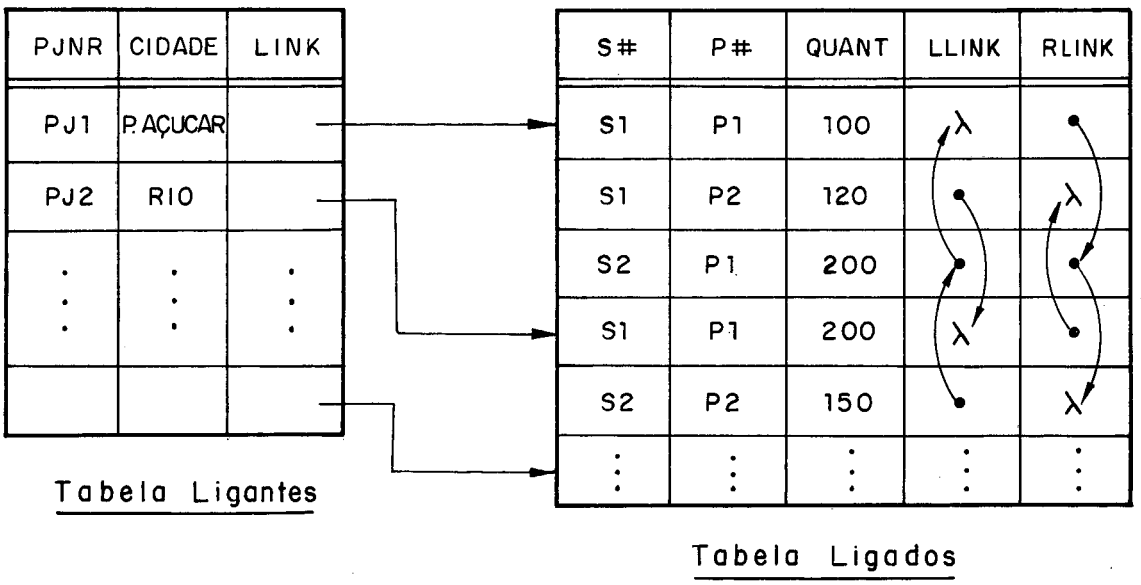


Figura IV.5 - Esquematização gráfica da representação de uma Tabela Ligacional através de duas relacionais.

Os atributos LINK, RLINK e LLINK mostrados na figura IV.5 são "internos" e portanto não são vistos pelo usuário. Como falado no capítulo II, os ponteiros internos são do tipo TID ("tuple identifier"), e portanto esse é o tipo dos atributos LINK, RLINK e LLINK concatenados às formas internas das tuplas de ligantes e ligados.

A tabela de ligantes será implementada como se fosse uma tabela relacional comum, e portanto poderá estar apontada por um índice em hashing de acordo com a chave primária especificada para a tabela ligacional associada. Aqui o cálculo (interno) do número de tuplas por página deverá levar em conta os 2 bytes extra gastos por cada tupla, devido à presença do campo LINK.

Devido à possibilidade da tabela básica que armazena a tabela dos ligados apresentar duas tuplas iguais (só que pertencentes a ligações diferentes), esta não possuirá "índi

ce em hashing" como acontece com a que armazena a de ligantes. Ao invés disso, as representações dos ligados de uma mesma ligação são mantidos duplamente encadeados, e esse encadeamento reflete a ordenação lógica entre eles de acordo com a chave especificada para a tabela ligada de cada ligação. Com isso, consegue-se um ganho razoável no tempo gasto para uma busca sem sucesso, uma vez que não é necessário ir sempre até o fim da cadeia.

É importante observar contudo, que todas essas adaptações no armazenamento devem ser completamente invisíveis ao usuário, e o SGBD é quem deve ser encarregado de toda a carga adicional de trabalho introduzida no armazenamento e recuperação de uma tabela ligacional.

Devido ao fato dos ligados de uma mesma ligação encontrarem-se logicamente ordenados, algumas precauções deverão ser tomadas na inserção e remoção de tuplas ligadas. A inserção de um ligado acarretará numa varredura a ser feita até se achar o lugar onde esta deverá ser logicamente feita. Depois de achado este lugar, a tupla é inserida fisicamente após a última tupla da tabela básica correspondente como normalmente, e os ponteiros são atualizados para refletir esta ordenação lógica. A remoção de uma tupla ligada exige que um cuidado extra seja tomado. Como a tabela de ligados é representada fisicamente através de uma tabela básica, como todas as outras aliás; a remoção de uma tupla normalmente implicará na compactação como falado no capítulo II. Neste momento, os ponteiros deverão ser atualizados de modo a refletir a nova posição da tupla movida e a nova ordenação dos ligados da ligação onde foi feita a retirada. Assim de maneira a tratar os ligados, novas primitivas

terão de ser adicionadas àquelas faladas na seção II.7. Seriam três essas primitivas:

a) LELIGDCHAV (ENTMEM1, DESLOC1, ENTMEM2, DESLOC2, POSRELLIGT, ENTMEM, DESLOC)

Objetivo - ler uma tupla ligada de acordo com o valor que ela apresenta para o(s) atributo(s) chave primária da tabela ligada, para uma ligação determinada.

b) INCLIGD (ENTMEM1, DESLOC1, ENTMEM2, DESLOC2, POSREL)

Objetivo - incluir uma tupla em uma tabela de ligados, para uma ligação determinada.

c) REMLIGD (ENTMEM1, DESLOC1, ENTMEM2, DESLOC2, TID, POSREL)

Objetivo - remover uma tupla de uma tabela de ligados, conhecendo-se a ligação a qual ela pertence.

A adoção dessa idéia para implementação, nos traz as seguintes vantagens e desvantagens:

a) Vantagens

. Acesso "direto" aos ligantes de acordo com a chave.

. Inclusão e Remoção de Ligados sem movimento em massa de tuplas.

. Adaptação quase direta ao Ambiente proposto.

b) Desvantagens

. Espaço gasto com ponteiros.

. Não contiguidade física dos ligados de uma mesma ligação.

Como idéia de otimização, aconselha-se uma política que tente aumentar a contiguidade física dos ligados, tentando manter próximos, de preferência nas mesmas páginas físicas, os ligados de uma mesma ligação. Isso pode ser feito normalmente no processamento, ou através de um utilitário a serviço do DBA.

IV.4. REPRESENTAÇÃO PARA ARQUIVOS

Os arquivos, como definido no capítulo III, são constituidos de uma tupla (sob nome FICHA) e de uma tabela relacional ou ligacional (sob nome TR ou TL). A forma de representação interna para as tabelas relacionais e ligacionais foi apresentada na seção anterior, e fica faltando portanto definir a representação interna da FICHA e como essas duas representações (da FICHA e da tabela) são combinadas para representar internamente um Arquivo.

Como já foi visto no capítulo II, cada tabela básica presente ao sistema caracteriza-se por ser descrita por uma entrada no Diretório de Tabelas Básicas. Como as tabelas relacionais e ligacionais sempre serão representadas através de tabelas básicas como já falado, através da adição de algumas características sobre as tabelas conceituais às informações descrevendo as tabelas básicas, o Diretório de Tabelas Básicas pode

se transformar no Diretório de Tabelas.

Em virtude da FICHA em MICROLOBAN registrar apenas informações padrão mantidas automaticamente pelo SGBD sobre os arquivos correspondentes, esta pode ser considerada como parte da descrição destes arquivos, com a diferença que é acessível pelos usuários. Assim, com a adição das informações (padrão) contidas na FICHA para um arquivo à entrada que descreve a tabela que a este pertence, o Diretório de Tabelas passa a descrever também os Arquivos presentes ao sistema, e transforma-se no Diretório de Arquivos e Tabelas como mostrado no próximo capítulo.

IV.5. REPRESENTAÇÃO PARA A FOLHA

A FOLHA, como visto no capítulo III, é uma construção de tipo padrão (pretipo) constituida por quatro tabelas ligacionais, as quais diferentemente daquelas componentes de Arquivos, sob nome TR ou TL, são referidas pelos nomes CV-COERÊNCIA, CV-ACESSO, CV-USUÁRIO e CV-FONTE. Assim a FOLHA será internamente representada pelo conjunto das representações internas,e respectivas descrições no Diretório de Arquivos e Tabelas, dessas tabelas que a constituem.

IV.6. REPRESENTAÇÃO PARA O ACTRAB

Como falado no capítulo III, o ACTRAB é constituido de uma tupla (sob nome FICHA) de tipo padrão, da FOLHA como falado acima e de um conjunto finito de Arquivos Relacionais e Ligacionais.

Como acontece com a FICHA de Arquivos, a do ACTRAB também tem tipo padrão e as suas informações gerais a respeito de todo o ACTRAB são mantidas automaticamente pelo SGBD, ainda que acessíveis pelos usuários autorizados. Assim, de maneira a tratá-los semelhantemente, a FICHA do ACTRAB ficará armazenada junto com a descrição do próprio Diretório de Arquivos e Tabelas, uma vez que é através deste que o ACTRAB é descrito, juntamente com todas as demais tabelas e arquivos.

Assim o ACTRAB será internamente representado pelo conjunto das representações internas, e respectivas descrições no Diretório de Arquivos e Tabelas, de todos os Arquivos Relacionais e Ligacionais criados no momento, juntamente com as representações internas e descrições das tabelas que constituem a FOLHA, mais as informações da FICHA armazenadas juntas com a descrição do próprio Diretório de Arquivos e Tabelas.

IV.7. REPRESENTAÇÃO PARA AS CONSTRUÇÕES DO CANAL AUXILIAR

De maneira a manter a uniformidade no armazenamento e recuperação, as construções do Canal Auxiliar (AUX) serão armazenadas juntas e da mesma forma que as do Acervo de Trabalho (ACTRAB), ou seja, também farão parte da Base de Dados Armazenada. Com isso, quatro vantagens principais são imediatemente antevistas:

a) As primitivas de acesso podem ser aplicadas tanto à construções da Base de Dados quanto do Canal Auxiliar;
b) As descrições das construções presentes ao Canal Auxiliar ficam juntas com as descrições das da Base de Dados, dispensando assim a criação de novas tabetas de descrições de dados;
c) As construções do Canal Auxiliar são automaticamente reconstruidas quando a Base de Dados o é; e
d) O Ambiente Interno definido no capítulo II pode ser usado também para suportar o Canal Auxiliar.

Através de restrições semânticas feitas em MICROLOBAN, a construção auxiliar foi fixada como sendo uma "nominação" |Santos[12]|, e que os únicos componentes imediatos por ele aceitos são: itens, tuplas, ligações, tabelas relacionais, tabelas ligacionais e coleções de itens, todos sob nomes associados.

De maneira a registrar quais as construções presentes à construção Auxiliar a cada instante, uma tabela será mantida pe

lo sistema. Essa tabela, mostrada na figura IV.6, é na verdade componente do Esquema Interno uma vez que a sua função é descrever dados. Assim ela deveria ser abordada no próximo capítulo já que a função deste é descrever o Esquema Interno; porém o será aqui de modo a facilitar a compreensão da representação interna do Canal Auxiliar.

TABCONSAUX

NOME	DPRETIPO	DTIPO	PONTEIRO
< nome >	< dpretipo >	< dtipo >	< ponteiro >
•	•	•	•
•	•	•	•
•	•	•	•

Figura IV.6 - Formato da Tabela de Construções do Canal Auxiliar (TABCONSAUX).

Obs.: . <nome> é o nome pelo qual a construção é reconhecida. Como o AUX foi considerado como uma nominação, todos os seus componentes imediatos tem que ter nome;

. <dpretipo> é a designação do pretipo da construção. Como falado, pode ser ITEM, TUPLA, LIGAÇÃO, TAREL, TALIG ou COLITENS;

. <dtipo> é a designação do tipo da construção em questão. Como falado acima, as descrições das construções do Canal Auxiliar ficam armazenadas juntas com as do ACTRAB e são identificados pela designação do tipo (DTIPO).

. <ponteiro> aponta para o "poço de itens" (próximo capítulo), se DPRETIPO = ITEM.

Uma tabela relacional ou ligacional (DPRETIPO = TAREL/TALIG) é internamente representada exatamente como essas do ACTRAB, ou seja, através de uma (se relacional) ou duas (se ligacional) tabelas básicas, com a diferença que uma entrada é criada para ela em TABCONSAUX de maneira a que se possa diferenciá-la das do ACTRAB. Essa entrada é quem a caracteriza como componente imediato da Construção Auxiliar.

Cada tupla ou ligação incluida como componente imediato da Construção Auxiliar será armazenada como se fosse uma tabela com uma única tupla/ligação, de maneira a poder ser armazenada através de tabelas básicas. O nome associado à tupla/ligação é assumido como nome da tabela e é quem aparece como

<nome> em TABCONSAUX. O <dpretipo> é TUPLA/LIGAÇÃO de modo a informar que apesar de armazenada como tabela a construção é uma tupla/ligação e <dtipo> é a designação de tipo da tupla/ligação.

As coleções de itens presentes à Construção Auxiliar também e pela mesma razão serão armazenadas como se fossem tabelas relacionais, com tuplas de um único atributo. O nome a ser associado ao atributo é padrão (ex. "item") já que a coleção de itens não apresenta nomes para os itens, e o nome associado à coleção é o que aparecerá no atributo NOME de TABCONSAUX.

Os itens componentes imediatos da Construção Auxiliar serão armazenados em uma construção chamada "poço de itens" e serão aí apontados pelo valor apresentado para PONTEIRO (TID), na entrada correspondente em TABCONSAUX. Toda vez que um item é inserido em AUX, o seu valor é colocado no "poço" e é estabelecido um ponteiro da entrada correspondente em TABCONSAUX para o "poço". O "poço de itens" será explicado no próximo capítulo.

A rotina de "avaliação de <end ponto>" |Santos[12], D'Albuquerque[2]| deve levar em conta as mudanças estruturais faladas acima, de maneira a garantir o acesso às informações armazenadas, sem que o usuário precise se preocupar com essas mudanças internas.

IV.8. REPRESENTAÇÃO PARA AS CONSTRUÇÕES DA ZONA INTERMEDIÁRIA

Como no Canal Auxiliar, as construções obtidas na Zona Intermediária também sofrerão mudanças estruturais de forma a serem suportadas pelo Ambiente Interno proposto no capítulo II.

Em virtude das construções que se pode obter na Zona Intermediária serem as mesmas que se pode armazenar no Canal Auxiliar, ou seja, tabelas relacionais e ligacionais, coleções de itens, tuplas, ligações e itens; elas serão armazenadas da mesma forma como falado na seção anterior, só que com duas diferenças:

a) Não há uma tabela que informe qual as construções presentes na Zona Intermediária pois isso é feito através de uma pilha da máquina virtual MICROLOBAN e é controlado pelo interpretador;

b) Os itens não serão colocados no "poço de itens" , mas sim nas pilhas da máquina virtual e é o interpretador que controla o seu armazenamento (como resultado do <obter construção> |Santos[12]|) e remoção ao fim do comando.

A máquina virtual e o tradutor estão fora do escopo desta tese, porém encontram-se detalhadamente explicados em |D'Albuquerque[2]|.

CAPÍTULO V

V. ESTRUTURA DO ESQUEMA INTERNO MICROLOBAN

Como pode-se ver em |Date[22]|, a finalidade do Esquema Interno é descrever a Visão Interna dos dados, a qual por toda essa tese temos chamado de Base de Dados Armazenada; não apenas no que diz respeito as estruturas de dados armazenadas, mas também aos índices existentes sobre estas, aos vários tipos de registros por elas armazenados, à representação para os campos armazenados, etc.

Na seção II.4 nós mostramos como o *Núcleo Básico* proposto poderia ser descrito, ou seja, apresentamos a parte do esquema interno dirigida especificamente para o *Núcleo Básico*. Assim neste capítulo nós completaremos a apresentação do Esquema Interno, agora em função de toda a arquitetura, e não apenas do *Núcleo Básico*.

Mais uma vez de maneira a manter a uniformidade no armazenamento e recuperação, o esquema interno será implementado através de tabelas relacionais e ligacionais exatamente como essas de nível conceitual e portanto armazenadas através de tabelas básicas. Com isso consegue-se não apenas que este seja armazenado junto e da mesma forma que os dados por eles descritos, como também que seja acessado e manipulado pelas mesmas primitivas e principalmente que também seja reconstruido quando estes o são.

V.1. DIRETÓRIO DE ARQUIVOS E TABELAS (DIRARQTAB)

É usado para descrever todas as tabelas básicas presentes à Base de Dados Armazenada, além de conter informações conceituais descrevendo as construções por estas armazenadas.

Em virtude de ser ele próprio considerado pelo SGBD como sendo uma tabela relacional (figura V.1), e portanto armazenado através de uma tabela básica, o que acontece é que ele se auto-descreve.

A figura V.1 mostra a representação gráfica para o Diretório de Arquivos e Tabelas onde pode-se notar que todas as informações mostradas na figura II.5 (relativas ao Ambiente proposto) foram mantidas, e que as demais informações agora contidas dizem respeito à estrutura de dados conceitual correspondente.

A seguir explicaremos a utilidade de alguns dos atributos mostrados na figura V.1, deixando outros sem explicação devido a serem auto-explicativos.

. <nome> é o nome associado à tabela/arquivo. Se é um arquivo ligacional, a entrada correspondente à tabela de ligantes recebe o nome do arquivo e a entrada correspondente à tabela de ligados recebe este mesmo nome antecedido pela letra T. (Exemplo: ACOMP→TACOMP).

. <tipo> informa qual o "tipo interno" da tabela, ou seja: 0 - Relacional
1 - De ligantes
2 - De ligados

. <dtipo> é a designação do tipo associado à tabela;

. <primentip> é a entrada onde começa a área blocada para a tabela no Índice de Páginas;

. <primentie> indica onde começa a área blocada para a tabela no Índice de Espalhamento. Se contiver um valor inválido a tabela não possui entradas alocadas no Índice de Espalhamento.

O valor <indac> informa qual a proteção e/ou autorização que o usuário corrente tem sobre o arquivo ou tabela correspondente. Esse campo é preenchido pela instrução <abrir acset> quanto à autorização e pela <estabelecer proteção> quanto à proteção. Para isso o byte reservado será dividido em dois meio bytes, o primeiro representando a proteção e o segundo a autorização. Assim temos as tabelas de valores mostradas abaixo:

Tabela de Valores de Autorização

0 → Arquivo não autorizado;

1 → Autorizado para consulta;

2 → Autorizado para modificação; e

3 → Autorizado para gerência.

Tabela de Valores de Proteção

0 → Não protegido;

1 → Protegido para leitura;

2 → Protegido para alteração; e

3 → Protegido para alteração porém sem direito de acesso imediato, ou seja, protegido por conexão.

Assim o atributo Indicação de Acesso é usado pelo SGBD para controlar a autorização e acesso à Base de Dados.

O valor <limite> informa o número de comparações que devem ser feitas para se concluir que uma determinada tupla não se encontra na tabela, de acordo com o método de hashing proposto na seção II.3, se é que esta tem associado um índice em hashing. A sua utilidade pode ser melhor entendida em |Souza[15]| .

Os atributos Deslocamento da Chave e Tamanho da Chave também são utilizados pela rotina de busca/inserção de acordo com o valor da chave primária.

NOME DO ARQUIVO/ TABELA	TIPO DA TABELA	DTIPO	CARDINAL CORRENTE	CARDINAL MAXIMA	TAMANHO DA TUPLA	1- ENTR. NO INDPAG	1- ENTR. NO INDESP	DESLOC. DA CHAVE	TAMANHO DA CHAVE	LIMITE ATUAL	INDICAÇÃO DE ACESSO	DATA DA CRIAÇÃO	DATA DA ULTIMA ATUALIZ.
<nome>	<tipo>	<dtipo>	<cardcor>	<cardmax>	<tamanho>	<primentip>	<primentie>	<desloc>	<tamchav>	<limite>	<indac>	<datcri>	<datat>
11 bytes	1 byte	10 bytes	2 bytes	2 bytes	1 byte	2 bytes	2 bytes	1 byte	1 byte	1 byte	1 byte	6 bytes	6 bytes

TOTAL - 47 bytes

Figura V.1 - Representação gráfica para o Diretório de Arquivos e Tabelas.

V.2. TABELA DE AVALIAÇÃO DE ENDEREÇO DE PONTO (TABAVALEND)

É usada pelo SGBD para controlar o procedimento de avaliação de endereço de ponto.

A Tabela de Avaliação de Endereço de Ponto será implementada através de uma tabela ligacional onde cada ligação informa os pontos escolhidos pela avaliação de um determinado critério ou os pontos a serem examinados na avaliação do critério corrente, em um endereço de ponto. O ligante contém informação gerais sobre os pontos tais como: pretipo, tipo, arquivo ao qual pertencem, se são pontos escolhidos ou candidatos, etc. A tabela ligada registra os endereços (TID) das construções nos pontos escolhidos/candidatos para o critério.

Devido ao escopo do endereço de ponto, a tabela sempre estará vazia após o término de um comando.

A tabela de Avaliação de Endereço de Ponto é detalhadamente abordada em |D'Albuquerque[2]|

V.3. TABELA DE MARCAS (TABMARCA)

É responsável por manter o sistema informado das marcas presentes ao Acervo e dos pontos por estas marcadas.

A tabela de marcas será implementada através de uma tabela ligacional onde cada ligação armazena informações sobre uma marca. O ligante registra informações gerais sobre a marca tal como o nome associado, a identificação do arquivo marcado, o pretipo da construção no ponto marcado, etc. A tabela de ligados registra os endereços (TID) das construções nos pontos marcados.

De modo a dar um tratamento único, a tabela de marcas também será usada para registrar as marcas implícitas introduzidas automaticamente pelo SGBD, ou seja, as marcas que definem um conjunto controlador para o comando <fazer para> e para as expressões PARA TODO, EXIST, JUNT e LIGA |Santos[12]|.

É importante observar contudo que uma marca implícita é automaticamente removida após o comando ou expressão que a originou e que as explícitas têm validade apenas na sessão corrente, o que implica que a tabela de marcas sempre estará vazia quando uma sessão é aberta.

V.4. TABELAS DA FOLHA CODIFICADA

Como apresentado no capítulo passado, os verbetes de coerência serão armazenados em CV-COERÊNCIA na forma de texto-fonte. Devido a dificuldade de interpretá-las dessa forma é que introduziu-se o que se chamou de Folha Codificada função é armazenar os verbetes de coerência numa forma mais dirigida para o SGBD, de forma a que o mesmo possa fazer as verificações de tipo. Na verdade essa é uma forma traduzida e reduzida da forma fonte dos verbetes de coerência, a qual é fornecida juntamente com a fonte pelo analisador, além dos tipos deduzidos a partir destes para as construções obtidas na Zona Intermediária e Canal Auxiliar. Também fazem parte da Folha codificada todos os tipos padrão predefinidos (pretipos) e os tipos associados às construções do próprio Esquema Interno.

A Folha Codificada é constituida de seis tabelas as quais representam tanto a composição quanto a conexão dos dados.

V.4.1. TABELA DE COMPOSIÇÃO DE NOMINAÇÕES (TABCOMPNOM)

É usada para armazenar a forma intermediária de todos os verbetes de coerência que definem tipos de nominação. Será implementada através de uma tabela ligacional como mostrado na figura V.2.

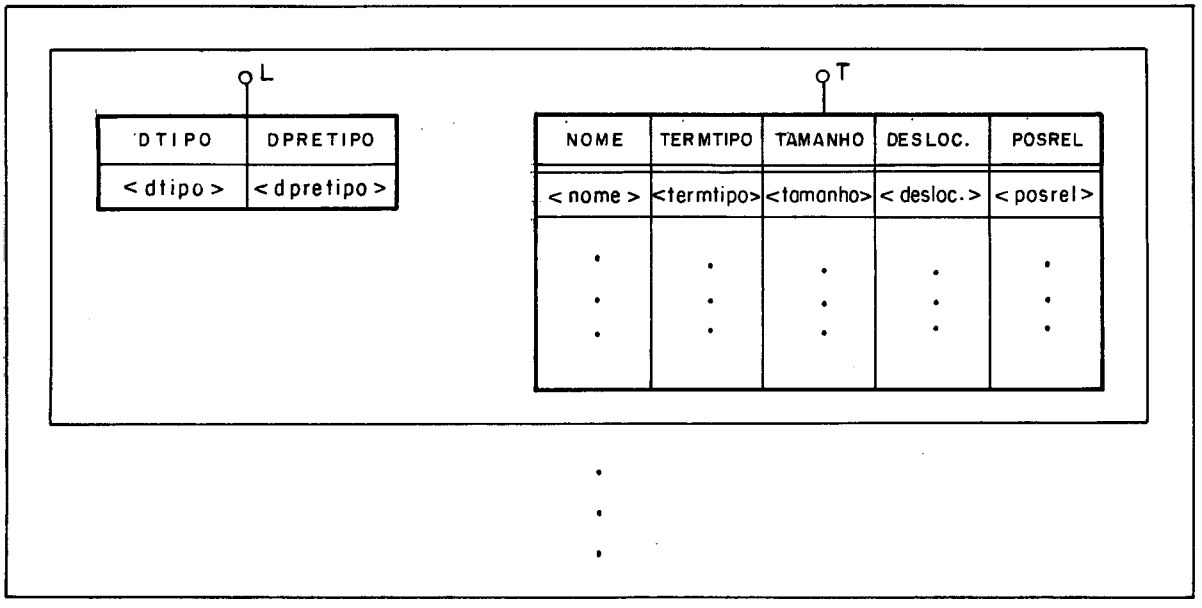


Figura V.2 - Representação gráfica da Tabela de Composição de Nominações.

Obs.: . <dtipo> é a designação do tipo em questão. É chave primária para esta tabela;

. <dpretipo> indica o pretipo da nominação, ou seja, ACTRAB, AREL, ALIG, TUPLA ou LIGAÇÃO;

. <nome> é o nome dado ao componente imediato. É chave da tabela ligada;

. <termtipo> informa a designação do tipo do componente imediato;

. <tamanho> é o tamanho (em bytes) do componente imediato;

. <desloc> é o deslocamento (em bytes) da construção. Só tem sentido se DPRETIPO = TUPLA; e

. <posrel> é a posição relativa do componente dentro da nominação. Também só tem sentido para tupla.

V.4.2. TABELA DE CONSISTÊNCIA DE COLEÇÕES (TABCONSCOL)

É usada para armazenar a forma intermediária de todos os verbetes de coerência que definem tipos de coleções. Ela será implementada através de uma tabela relacional como mostrado na figura V.3.

DTIPO	DPRETIPO	TERMTIPO	DESLCHAV	TAMCHAV	CARDINAL
< dtipo >	< dpretipo >	< termtipo >	< deslchav >	< tamchav >	< cardinal >

Figura V.3 - Representação gráfica para a tabela de Coerência de Coleções.

Obs.: . <dtipo> informa a designação do tipo em questão. É chave primária para a tabela;

. <dpretipo> informa o pretipo da coleção, ou seja, TAREL, TALIG ou COLITENS;

. <termtipo> informa a designação do tipo do componente imediato da coleção;

. <deslchave> informa o deslocamento de início da chave primária na tupla ou ligante;

. <tamchave> informa o tamanho em bytes da chave primária associada ao tipo em questão; e

. <cardinal> informa a cardinalidade máxima prevista para o tipo em questão.

V.4.3. TABELA DE DEFINIÇÃO DE SIGLAS (TABDEFSIG)

Registra os tipos de siglas previstas e por quantos caracteres cada uma é composta.

Todas as construções do pretipo SIGLA devem ter um tipo definido nesta tabela, a qual será implementada através de uma tabela relacional como pode-se ver na figura V.4.

DTIPO	TAMANHO
< dtipo >	< tamanho >
.	.
.	.
.	.

Figura V.4 - Representação gráfica para a Tabela de Definição de Siglas.

Obs.: . <dtipo> é a designação do tipo de SIGLA em questão. É chave primária da tabela; e

. <tamanho> informa o número de caracteres que compõem as representações das ocorrências do tipo de SIGLA.

V.4.4. TABELA DE DEFINIÇÃO POR EXTENSÃO (TABDEFEXT)

Registra os tipos de construção cuja coerência é definida através da enumeração de todos os valores possíveis de ocorrerem, ou seja, o domínio é definido por extensão. Esta tabela será implementada em forma de tabela ligacional, como pode-se ver na figura V.5.

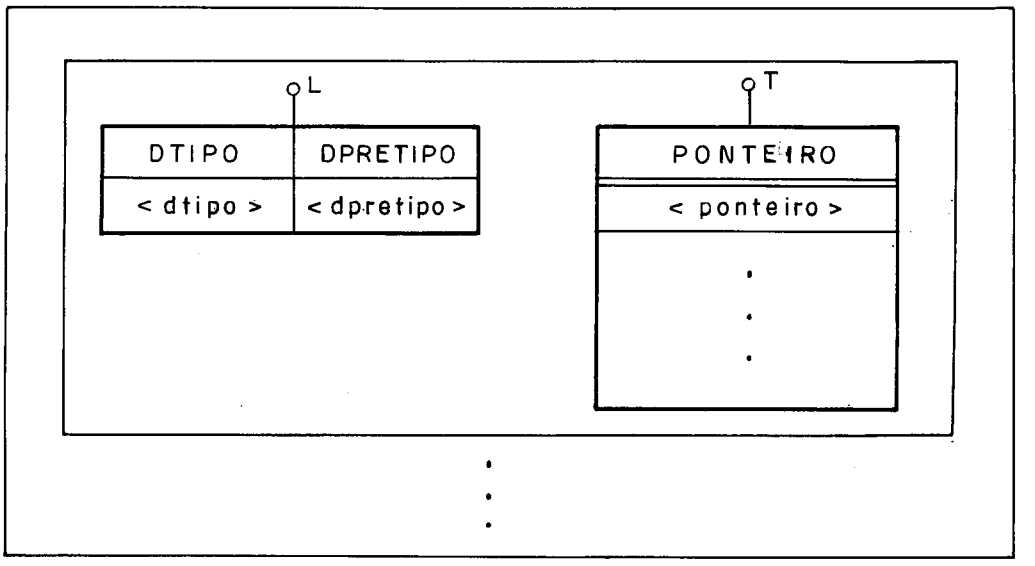


Figura V.5 - Representação gráfica para a Tabela de Definição por Extensão.

Obs.: . <dtipo> é a designação do tipo de construção. É chave primária para a tabela.

. <dpretipo> informa o pretipo associado à construção. Pode ser DATA, HORA, REAL, INT e SIGLA;

. <ponteiro> aponta para a entrada que contém o valor correpondente no Poço de Itens o qual será daqui a pouco definido. Cada uma das ocorrências do valor <ponteiro> aponta para um dos valores possíveis para a construção no Poço de Itens.

V.4.5. TABELA DE DEFINIÇÃO POR INTERVALO (TABDEFINT)

Define intervalos onde os valores de um determinado tipo podem ocorrer. Será implementada através de uma tabela relacional como esta mostrada na figura V.6.

DTIPO	DPRETIPO	VALOR 1	OPERADOR 1	VALOR 2	OPERADOR 2
< dtipo >	< dpretipo >	< valor 1 >	< operador1 >	< valor 2 >	< operador2 >
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.

Figura V.6 - Representação gráfica para a Tabela de Definição por Intervalo.

Obs.: . <dtipo> identifica a designação do tipo. É chave primária para a tabela;

. <dpretipo> informa o pretipo associado ao item que se está definido. Pode ser INT, REAL, DATA e HORA;

. <valor 1> e <valor 2> são ponteiros para o Poço de Itens, para os valores que delimitam o intervalo do tipo sendo definido; e

. <operador 1> e <operador 2> são os operadores relacionais a serem aplicados sobre os valores apontados por <valor1> e <valor2> de modo a determinar o intervalo.

V.4.6. TABELA DE CONEXÃO DO ACTRAB (TABCONACT)

Registra as conexões entre os arquivos que compõem o ACTRAB. Essas conexões refletem os "relacionamentos" entre os arquivos, através de atributos de domínio comum.

A conexão entre dois arquivos caracteriza a exigência que o conjunto de valores ocorrendo para um atributo em uma tabela de um arquivo, tem que ser subconjunto do conjunto de valores ocorrendo para outro atributo na tabela do outro arquivo, tendo os dois atributos domínio comum.

A Tabela de Conexão do ACTRAB será implementada como sendo uma tabela relacional como essa mostrada na figura V.7.

IDENT. DO ARQ 1	ATRIBUTO 1	IDENT. DO ARQ 2	ATRIBUTO 2
< arq 1 >	< atributo 1>	< arq 2 >	< atributo 2 >

Figura V.7 - Representação gráfica para a Tabela de Conexão do ACTRAB.

Obs.: . <arq 1> identifica o arquivo no qual o conjunto de valores ocorrendo para ATRIBUTO1 tem que ser subconjunto do conjunto de valores ocorrendo para ATRIBUTO2 do arquivo identificado por <arq 2 >

. ATRIBUTO1 e ATRIBUTO2 têm que ter o mesmo domínio, ou seja, serem do mesmo tipo.

V.5. POÇO DE ITENS

Será constituido de seis tabelas relacionais sendo cinco tabelas de valores, uma para cada um dos pretipos que constituem o pretipo ITEM; e uma tabela de hashing cuja função é fazer o mapeamento entre os ponteiros para o poço e os valores correspondentes nas tabelas de valores. Com isso qualquer acesso ao poço é sempre feito através da tabela de hashing.

Todas as tabelas de valores têm o mesmo formato, diferindo apenas no número de bytes do atributo VALOR já que este é determinado pelo número máximo de bytes associado a cada um dos pretipos.

A figura V.8 representa graficamente uma tabela de valores.

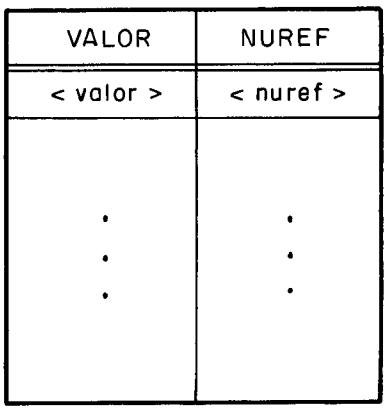


Figura V.8 - Representação gráfica de uma tabela de valor.

Obs.: . <valor> é o valor apresentado pelo item em questão. O número de bytes que o constitui varia de tabela para tabela; e

. <nuref> é o número de referências feitas ao valor. Essas referências podem estar em TABCONSAUX, TABDEFINT e TABDEFEXT .

A tabela de hashing como falado é quem se encarrega dos acessos às tabelas de valores. Duas são as razões principais do seu uso:

a) Tornar mais rápido o procedimento de conclusão se um item já se encontra ou não no poço; e

b) Evitar que as tabelas que apontam para o poço sejam afetadas pelas mudanças de posição das tuplas das tabelas básicas que representam internamente as tabelas de valores.

De maneira a servir como índice em hashing para as tabelas de valores, a tabela de hashing será blocada e terá todas as suas entradas alocadas no momento da sua criação, da mesma forma como o Índice de Espalhamento apresentado no capítulo II. Na verdade a tabela de hashing pode ser imaginada como o "índice de espalhamento do poço de itens", e só não será implementada através do Índice de Espalhamento devido aos rearranjos que são feitos nas entradas deste no tratamento de colisões, o que impossibilita a razão b anteriormente citada. Assim a tabela de hashing será implementada da mesma forma que o Índice de Espalhamento, só que o método de tratamento de colisões associado não faz os rearranjos no índice.

A figura V.9 ilustra a Tabela de Hashing.

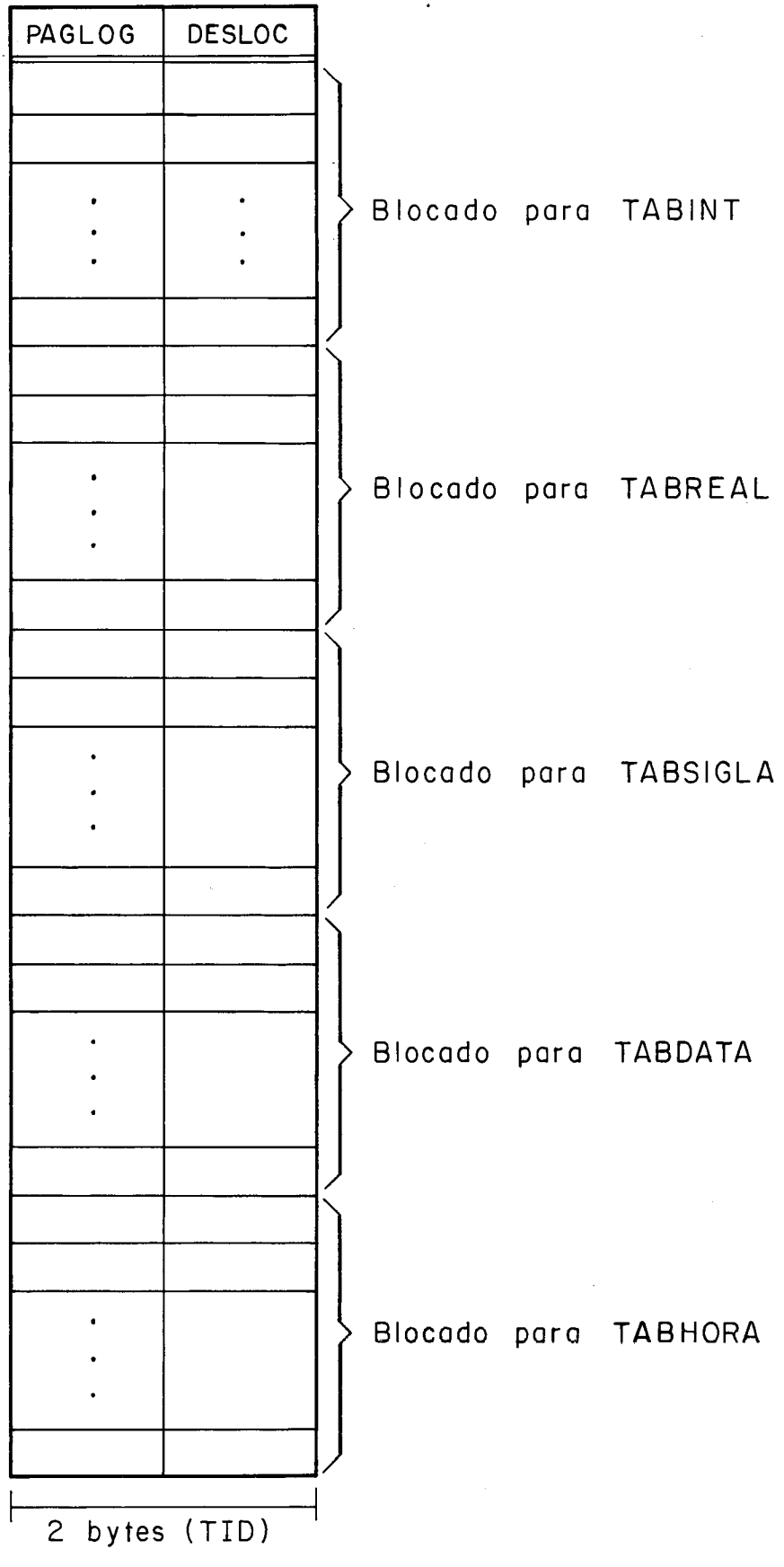


Figura V.9 - Representação gráfica para a Tabela de Hashing.

CAPÍTULO VI

VI. CONCLUSÕES

Em virtude do enfoque geral dado à proposta de Ambiente Interno, achamos que o mesmo pode ser adotado com relativa facilidade por outras implementações de SGBD. Claro que algumas mudanças deverão ser feitas, principalmente em função da interface a ser suportada e da máquina escolhida para implementação, porém mantendo-se a filosofia básica.

Também gostariamos de destacar o alto grau de padronização conseguido pela implementação MICROLOBAN não apenas com respeito ao acesso e manipulação, mas principalmente na reconstrução dos dados, em virtude de termos adotado um único esquema de armazenamento/recuperação para as estruturas de dados conceituais (Base de Dados, Canal Auxiliar e Zona Intermediária) e as internas controladas pelo SGBD.

Como próxima etapa deste trabalho, pretende-se a implementação do Ambiente Interno proposto no capítulo II no computador COBRA-300 de forma a que testes de funcionamento possam ser realizados e que se possa chegar a algumas conclusões sobre o desempenho e a partir daí tentar promover correções e refinamentos em pontos que aparecem mais críticos, e dar à proposta uma forma final com base nas observações e alterações feitas.

A partir daí então pretende-se a aplicação prática da proposta não apenas na implementação MICROLOBAN, mas também a outras implementações tais como COPPEREL e SIBANDABAS, de forma a que novas observações possam ser feitas e refinamentos a partir daí promovidos, agora não apenas a nível interno, mas também a nível conceitual e em relação à arquitetura e à interface como um todo.

ANEXO

ANEXO

Neste anexo é apresentada a especificação das principais *primitivas de acesso* a serem suportadas pela implementação MICROLOBAN de forma a promover a comunicação entre os níveis conceitual e interno.

Essas primitivas encontram-se distribuidas em três grandes categorias, distribuição esta feita em função da parte do Ambiente Interno na qual elas atuam. São as seguintes estas categorias:

A - Primitivas de Gerência da Memória;

B - Primitivas de Acesso e Manipulação das Tabelas Básicas; e

C - Primitivas de Reconstrução.

A.1. OBTENTPAG (INDCRI, NUPAGFIS, PRIORI, ENTMEM)

Objetivo - Devolver a entrada (0 a N-1) na Memória de Páginas onde se encontra a página física de número NUPAGFIS, carregando-a ou criando-a se ela não se encontra na memória.

Parâmetros:

- INDCRI (E) - informa se a página já está gravada no disco (0) ou se deve ser criada (1);
- NUPAGFIS (E) - informa o número da página física desejada;
- PRIORI (E) - informa a "prioridade de permanência na memória" que se quer dar à página;
- ENTMEM (S) - devolve a entrada na memória para onde a tupla foi lida.

Procedimento:

```
1 - Se VETPRI (2 * INIFILA + 1) ≥ N;
    então fazer
            FIMFILA ← (FIMFILA + 1) MOD N;
            ENTMEM  ← VETPRI (2 * FIMFILA + 1) - N; I ← FIMFILA;
            vá para 6;
          end;
```

```
2 - I ← INFILA;
    CONTINUA ← TRUE;
    ACHADO   ← FALSE;

3 - Enquanto CONTINUA
    fazer
      Se MEMORIA (VETPRI (2 * I + 1)) = NUPAGFIS
      então  fazer
               CONTINUA ← FALSE;
               ACHADO   ← TRUE;
             end
      senão  fazer
               Se I = FIMFILA
               então CONTINUA ← FALSE
               senão I ← (I + 1) MOD N;
             end;
    end;

4 - Se ACHADO
    então  fazer
             ENTMEM ← VETPRI (2 * I + 1);
             Se PRIORI ≠  N-1
             então REARRANJA (I, PRIORI, J)
             senão  fazer
                      Se I ≠ FIMFILA
                      então VETPRI (2*I+1) ↔ VERPRI(2*((I+1)MOD N)+1)
                    end;
             vá  p/ FIM;
           end;
```

```
5 - FIMFILA ← (FIMFILA + 1) MOD N; I ← FIMFILA;
    Se  FIMFILA = INIFILA
    então  fazer
             PAGE-OUT (ENTMEM);
             INIFILA ←  (INIFILA + 1) MOD N;
             Se  PRIORI ≠  N-1 então REARRANJA(FIMFILA,PRIORI,I);
           end
    senão  fazer
             ENTMEM ← VETPRI (2 * FIMFILA + 1) - N;
             Se PRIORI ≠ N-1 então REARRANJA (FIMFILA,PRIORI,I);
           end;

6 - Se INDCRI = 1
    então  fazer
             MEMORIA (ENTMEM * 255) ← NUPAGFIS;
             MEMORIA (ENTMEM * 255 + 2) ←"nº do comando corrente";
             VETPRI (2 * I) ← 2;  ÷ alterada
           end
    senão  fazer
             Chamar LEREGSOM (NUPAGFIS, ENTMEM, ERRO);
             VETPRI (2 * I) ← 0;
           end;
```

A.2. PAGE-OUT (ENTMEM)

Objetivo - Liberar uma entrada na Memória de Páginas, com base na "prioridade de permanência".

Parâmetros:

. ENTMEM - informa o número da entrada liberada (0 a (E) N-1) pela devolução de uma página física ao disquete.

Procedimento:

```
1 - I ← INIFILA;
    CONTINUA ← TRUE
    Enquanto CONTINUA
    fazer
      Se VETPRI (2 * I) = 0 ou 2
      então CONTINUA ← FALSE
      senão I ← (I + 1) MOD N;
    end;

2 - REARRANJA (I,0,J);
    Se VETPRI (2 * INIFILA) = 2
    então  fazer
             ENTMEM ← VETPRI (2 * INIFILA + 1);
             NUPAG  ← MEMORIA (ENTMEM  * 255);
             chamar GRVREGSOM (UD, ENTMEM, NUPAG);
           end;
```

A.3. REARRANJA (ENTORIG, PRIORDEST, ENTDEST)

Objetivo - Rearranjar a fila de prioridades.

Parâmetros:

- . ENTORIG - (E) informa a entrada no vetor de prioridades onde se encontra o apontador para a página na cuja prioridade se quer mudar;
- . PRIORDEST - (E) informa a prioridade que se quer dar à página apontada por ENTORIG; e
- . ENTDEST - (S) retorna a nova entrada que aponta para a página.

Procedimento:

```
1 - ENTDEST ← (INIFILA + PRIORDEST) MOD N;
    Se ENTORIG = ENTDEST vá p/ FIM;
    L ← INIFILA;
    PRIORORIG ← 0;
    CONTINUA  ← True;
    Enquanto CONTINUA
    fazer
      Se L = ENTORIG  então  CONTINUA ← False
                      senão  fazer
                               L ← (L + 1) MOD N;
                               PRIORORIG ← PRIORORIG + 1;
                             end
    end;
    CONTINUA ← true;
```

```
2 - Se PRIORORIG < PRIORDEST
    então  fazer
             Se VETPRI (2*ENTDEST+1)≥ N então ENTDEST ← FIMFILA;
             SALVA ← VETPRI (2 * ENTORIG + 1);
             L ← ENTORIG;
             Enquanto CONTINUA
             fazer
               Se L = ENTDEST
               então  fazer
                        CONTINUA ← False;
                        VETPRI (2 * L + 1) ← SALVA;
                      end
               senão  fazer
                        VETPRI (2 * L + 1) ← VETPRI(2*((L+1)MOD N)+1);
                        L ← (L + 1) MOD N;
                      end;
             end;
           end
    senão  fazer
             SALVA ← VETPRI (2 * ENTORIG + 1);
             L ← ENTDEST;
             Enquanto CONTINUA
             fazer
               Se L = ENTORIG
               então  fazer
                        CONTINUA ← False;
                        VETPRI (2 * L + 1) ← SALVA;
                      end
               senão  fazer
                        SALVA1 ← VETPRI (2 * L + 1)
                        VETPRI (2 * L + 1) ← SALVA;
                        SALVA ← SALVA1;
                        L ← (L + 1) MOD N;
                      end ;
             end;
           end;
```

A.4. COPIAR (ENTMEM1, DESLOC1, ENTMEM2, DESLOC2, TAMANHO)

Objetivo - copiar a informação de TAMANHO bytes a partir da posição DESLOC1 da página ENTMEM1 para a página ENTMEM2 a partir da posição DESLOC2.

Parâmetros:

. ENTMEM1, DESLOC1 - endereçam o início da informação
(E) a ser copiada;

. ENTMEM2, DESLOC2 - endereçam o início da área onde
(E) a informação será copiada;

. TAMANHO - informa o tamanho da informação a ser co
(E) piada.

Procedimento:

1 - Se MEMORIA (ENTMEM2 * 255) + 2) ≠ "número do comando"
então fazer
-REGCORREC ← REGCORREC + 1;
-MEMORIA ((ENTMEM2 * 255) + 2) ← "número do comando"
-Chamar GRVREGSOM (UD,ENTMEM2,REGCORREC), a qual gravará a cópia da página a ser alterada na cadeia de reconstrução.
end;

```
2 - Para I ← 0, incrementando de 1, até TAMANHO - 1
    fazer
      -MEMORIA ((ENTMEM2 * 255) + DESLOC2 + I)  ←
                    MEMORIA ((ENTMEM1 * 255) + DESLOC1 + I);
      -L ← INIFILA;
      -CONTINUA ← True;
      -Enquanto CONTINUA
       fazer
         Se  VETPRI (2 * L + 1) = ENTMEM2
         então  fazer
                  -Se VETPRI (2 * L) = 0
                   então VETPRI (2 * L) ← 2
                   senão Se VETPRI (2 * L) = 1
                         então VETPRI (2 * L) ← 3;
                  -CONTINUA ← false;
                end
         senão  L ← (L + 1) MOD N;
       end;
    end;
```

Obs.: - REGCORREC indica o "registro corrente para gravação" no disquete de Reconstrução.

- UD indica a unidade do disquete da Reconstrução.

A.5. LIMPAMEM

Objetivo - Limpar a memória, ou seja, desocupar todas as entradas.

Procedimento:

```
1 - L ← INIFILA;
    Se VETPRI (2 * L + 1) ≥ N então vá para FIM;
    CONTINUA ← True;
    Enquanto CONTINUA
    fazer
      Se VETPRI (2 * L) = 2 ou 3 ; a página foi alterada
      então  fazer
               ENTMEM ← VETPRI (2 * L + 1);
               NUPAG  ← MEMORIA (ENTMEM * 255);
               chamar GRVREGSOM (UD, ENTMEM, NUPAG);
             end;
      VETPRI (2 * L + 1) ← VETPRI (2 * L + 1) + N;
      Se L = FIMFILA então CONTINUA ← False
                      senão L ← (L + 1) MOD N;
    end;
    INIFILA ← 0;
    FIMFILA ← N-1;
```

Obs.: - UD indica a unidade do disquete da BDA.

A.6. PREPMEM

Objetivo - preparar a memória de páginas para o uso.

Procedimento:

```
1 - Para I ← 0, incrementando de 1, até N-1
      fazer
        VETPRI (2 * I) ← 0
        VETPRI (2 * I + 1) ← N+I;
      end
      INIFILA ← 0;
      FIMFILA ← N-1;
```

B.1. LERPAGLOG (PRIMENT, NUPAGLOG, NUPAGFIS)

Objetivo - Devolver o número da página física correspondente à página lógica informada.

Parâmetros:

. PRIMENT - informa qual a primeira entrada no índice
(E) de páginas para a tabela que contém a página desejada;

. NUPAGLOG - Informa o número da página lógica desejada;
(E)

. NUPAGFIS - Retorna o número da página física correspondente à página lógica desejada.
(S)

Procedimento:

1 - Calcular a entrada no Índice de Páginas onde está a página lógica pretendida:

ENTRADA ← PRIMENT + NUPAGLOG;

2 - Calcular em que página lógica do próprio Índice de Páginas está a página lógica pretendida, e o seu deslocamento nesta:

PAGLOGIP ← ⌈ENTRADA/125⌉;

DESLOC ← ((ENTRADA-1) MOD 125) * 2 + 4;

3 - Chamar OBTENTPAG (0,1,PRIORI,ENTMEM) a qual devolverá a entrada na "Memória de Páginas" onde se encontra a página física 1, a qual contém as páginas lógicas que compõem o próprio Índice de Páginas.

4 - Obter o número da página física correspondente à página lógica PAGLOGIP:

NUPAGFISIP ← MEMORIA ((ENTMEM * 255) + (PAGLOGIP * 2) + 4);

5 - Chamar OBTENTPAG (0, NUPAGFISIP, PRIORI, ENTMEM) a qual devolverá a entrada na memória onde está a página física que contém a página lógica desejada;

6 - NUPAGFIS ← MEMORIA ((ENTMEM * 255) + DESLOC).

B.2. LETUPPOS (POSREL, TAMANHO, PRIMENT, ENTMEM, DESLOC)

Objetivo - Ler uma tupla de acordo com a sua posição relativa na tabela.

Parâmetros:

. POSREL - informa a posição relativa na tabela da (E) tupla desejada (1 → I);

. TAMANHO - informa o tamanho da tupla desejada; (E)

. PRIMENT - informa o valor da primeira entrada no (E) Índice de Páginas para a tabela onde se encontra a tupla a ser lida;

. ENTMEM - retorna a entrada na Memória de Páginas (S) onde se encontra a página com a tupla desejada;

. DESLOC - retorna o deslocamento da tupla na pági (S) na.

Procedimento:

1 - Calcular em que página lógica e com que deslocamento (em bytes) a tupla pretendida se encontra:

NUTUPS ← ⌊251/TAMANHO⌋; ÷ nº de tuplas por página

PAGLOG ← ⌈POSREL/NUPUS⌉; ÷ página lógica da tupla

DESLOC ← ((POSREL-1)MOD NUTUPS)*TAMANHO + 4; ÷ deslocamento

2 - Chamar LERPAGLOG (PRIMENT, PAGLOG, NUPAGFIS) a qual devolverá o "número da página física" correspondente à página lógica desejada;

3 - Chamar OBTENTPAG (0, NUPAGFIS, PRIORI, ENTMEM) a qual devolverá a entrada na Memória de Páginas onde se encontra a página com a tupla pretendida.

B.3. LETUPTID (TID, PRIMENT, ENTMEM, DESLOC)

Objetivo - ler uma tupla de acordo com o TID fornecido.

Parâmetros:

. TID - informa o TID para a tupla a ser lida; (E)

. PRIMENT - informa o valor para o campo "primeira (E) entrada em INDPAG" para a tabela da qual se quer ler a tupla;

. ENTMEM - retorna a entrada na Memória de Páginas (S) onde se encontra a página com a tupla lida;

. DESLOC - retorna o deslocamento da tupla na pá- (S) gina.

Procedimento:

1 - PAGLOG ← TID(1); ÷ página lógica com a tupla;
DESLOC ← TID(2); ÷ deslocamento da tupla na página.

2 - Chamar LERPAGLOG (PRIMENT, PAGLOG, NUPAGFIS), a qual devolverá o "número da página física" correspondente à página lógica informada no TID;

3 - Chamar OBTENTPAG (0, NUPAGFIS, PRIORI, ENTMEM), a qual devolverá a entrada na memória onde se encontra a página que contém a tupla pretendida.

B.4. ALOCPAGFIS (ENTMEM1, DESLOC1, NUPAGLOG, ENTMEM)

Objetivo - alocar uma nova página física para a tabela descrita pela tupla (ENTMEM1, DESLOC1).

Parâmetros:

. ENTMEM1, DESLOC1 - (E) endereçam na Memória de Páginas a tupla de DIRARQTAB que descreve a tabela para a qual se quer alocar a página física;

. NUPAGLOG - (E) informa o número da página lógica correspondente à nova página física;

. ENTMEM - retorna a entrada na Memória de Páginas on
(S) de se encontra a página alocada.

Procedimento:

1 - Chamar LETUPPOS (4,2 PRIMENT, ENTMEM, DESLOC) a qual devolverá a tupla do DIRARQTAB que descreve a PILHADISP.PRIMENT indica a "primeira entrada em INDPAG" para o DIRARQTAB e o seu valor é fixo e conhecido;

2 - Obter a cardinalidade corrente para a PILHADISP a partir da tupla lida:
CARDCOR ← MEMORIA ((ENTMEM * 255) + DESLOC + 22);

3 - Chamar LETUPPOS (CARDCOR, 2, PRIMENT1, ENTMEM2, DESLOC2) a qual devolverá a última tupla da PILHADISP com o número da página física a alocar. PRIMENT1 indica a "primeira entrada em INDPAG" para a PILHADISP e seu valor é fixo e conhecido;

4 - NUPAGFIS ← MEMORIA ((ENTMEM2 * 255) + DESLOC2);

5 - Diminuir 1 da cardinalidade corrente da PILHADISP:
MEMORIA (N * 255 + POSDISP) ← CARDCOR - 1;
COPIAR (N, POSDISP, ENTMEM, DESLOC, 2);

6 - Calcular a entrada em DIRARQTAB correspondente à página lógica correspondente à página física sendo alocada:
PAGABS ← MEMORIA ((ENTMEM1 * 255) + DESLOC1 + 27) +
NUPAGLOG;

7 - Chamar LETUPPOS (PAGABS, 2, PRIMENT2, ENTMEM, DESLOC),a qual devolve a tupla de DIRARQTAB correspondente à NUPAGLOG.
PRIMENT2 é fixo e conhecido.

8 - Fazer a ligação página-lógica → página física:
MEMÓRIA (N * 255 + POSDISP) ← NUPAGFIS;
COPIAR (N, 0, ENTMEM, DESLOC, 2);

9 - Chamar OBTENTPAG (1, NUPAGFIS, PRIORI, ENTMEM) a qual criará a página alocada na memória e devolverá a entrada onde ela foi criada.

B.5. DESALOCPAG (ENTMEM)

Objetivo - desalocar uma página física, ou seja, devolvê-la à Pilha de Disponíveis (PILHADISP).

Parâmetros:

. ENTMEM - informa a entrada na memória onde se encontra a página a liberar. (E)

Procedimento:

1 - Chamar LETUPPOS (4, 2, PRIMENT, ENTMEM1, DESLOC1) a qual devolverá a tupla de DIRARQTAB que decreve PILHADISP. PRIMENT é fixo e conhecido.

2 - Chamar INCTUP (ENTMEM1, DESLOC1, ENTMEM, Ø, RESULT) a qual incluira o número da página a liberar na PILHADISP;

3 - L ← INIFILA;
CONTINUA ← True;
Enquanto CONTINUA
fazer
Se VETPRI (2 * L + 1) = ENTMEM
então CONTINUA ← False
senão L ← (L + 1) MOD N;
end;
CONTINUA ← True; SALVA ← VETPRI (2 * L + 1) + N;
Enquanto CONTINUA

```
fazer
  Se L = FIMFILA
  então  fazer
           VETPRI (2 * FIMFILA + 1) ← SALVA;
           Se FIMFILA = 0 então FIMFILA ← N-1
                          senão FIMFILA ←FIMFILA - 1;
         end
  senão  fazer
           VETPRI (2 * L + 1) ← VETPRI (2 * ((L+1) MOD N)+1);
           L ← (L + 1) MOD N;
         end;
end;
```

B.6. INCTUP (ENTMEM1, DESLOC1, ENTMEM2, DESLOC2, RESULT)

Objetivo - incluir uma tupla em uma tabela básica.

Parâmetros:

. ENTMEM1, DESLOC1 (E) - endereçam na memória a tupla de DIRARQTAB que descreve a tabela básica onde será feita a inclusão;

. ENTMEM2, DESLOC2 (E) - endereçam na memória a tupla a ser incluida;

. RESULT (S) - retorna se a inclusão foi (0) ou não (1) realizada.

Procedimento:

1 - Obter algumas informações sobre a tabela onde será feita a inclusão, a partir da tupla de DIRARQTAB que a descreve:

```
CARDCOR ← MEMORIA ((ENTMEM1 * 255) + DESLOC1 + 22);
TAMANHO ← MEMORIA ((ENTMEM1 * 255) + DESLOC1 + 26);
PRIMENT ← MEMORIA ((ENTMEM1 * 255) + DESLOC1 + 27);
NUTUPS  ← ⌊251/TAMANHO⌋;
NUPAGS  ← ⌈CARDCOR/NUTUPS⌉;
```

2 - Se (CARDCOR MOD NUTUPS) = 0

```
então  fazer
         NUPAGS ← NUPAGS + 1;
         chamar ALOCPAGFIS (ENTMEM1, DESLOC1, NUPAGS, ENTMEM);
         DESLOC ← 4;
       end
senão  fazer
         chamar LERPAGLOG (PRIMENT, NUPAGS, NUPAGFIS);
         chamar OBTENTPAG (0, NUPAGFIS, PRIORI, ENTMEM);
         DESLOC ← (CARDCOR - ((NUPAGS - 1)*NUTUPS))*TAMANHO +
             4 + TAMANHO;
       end;
```

3. PRIMENTTE ← MEMORIA ((ENTMEM1 * 255) + DESLOC1 + 29);
   Se PRIMENTTE ≠ 0 então
   fazer -chamar LIGAHASH (ENTMEM1,DESLOC1,ENTMEM2,DESLOC2,
   NUPAGS,RESULT) a qual incluirá a chave da tupla sendo incluida no Índice de Espalhamento, ou retornará RESULT = 1 se a chave não pode ser incluida;
   -Se RESULT = 1 vá para FIM;
   end;

4. Chamar COPIAR (ENTMEM2,DESLOC2,ENTMEM,DESLOC,TAMTUP), a qual incluirá a tupla na última página física;

5. Aumentar a cardinalidade corrente;
   - MEMORIA (N*255 + POSDISP) ← CARDCOR + 1;
   - Chamar COPIAR (N,POSDISP,ENTMEM1,DESLOC1 + 22,2), a qual atualizará a cardinalidade corrente;

6. Verificar se a tupla incluida é um ligante, e neste caso fazer TID = 0:
   - TIPTAB ← MEMORIA ((ENTMEM1 * 255) + DESLOC1 + 11);
   - Se TIPTAB = 1 ; tabela de ligantes
     então fazer
     MEMORIA (N * 255 + POSDISP) ← 0;
     DESLOC ← DESLOC + TAMANHO - 2;
     COPIAR (N,POSDISP,ENTMEM,DESLOC,2);
     end;

B.7. LETUPCHAV (ENTMEM1, DESLOC1, ENTMEM, DESLOC, ENTMEM2, DESLOC2)

Objetivo - Ler uma tupla de acordo com o valor que ela apresenta para o(s) atributo(s) "chave-primária".

Parâmetros:

- . ENTMEM1 , DESLOC1 (E) - endereçam na memória a tupla do DIRARQTAB que descreve a tabela da qual se quer ler a tupla;
- . ENTMEM , DESLOC (E) - endereçam na memória o valor apresentado para a chave primária da tupla que se quer ler;
- . ENTMEM2 , DESLOC2 (S) - retornam a entrada na Memória de Páginas e o deslocamento para a tupla lida.

Procedimento:

1 - TAMCHAV ← MEMORIA ((ENTMEM1 * 255) + DESLOC1 + 32);
- DESCHAV ← MEMORIA ((ENTMEM1 * 255) + DESLOC1 + 31);
- chamar HASH (ENTMEM,DESLOC,TAMCHAV,INICIAL,SALTO), a qual devolverá o "home-address" e o "salto" para a chave fornecida;

2 - Obter algumas informações sobre a tabela a partir do DIRARQTAB:

- CARDCOR ← MEMORIA((ENTMEM1 * 255) + DESLOC1 + 22);
- TAMTUP ← MEMORIA((ENTMEM1 * 255) + DESLOC1 + 26);
- PRIMENTIE ← MEMORIA((ENTMEM1 * 255) + DESLOC1 + 29);
- LIMATUAL ← MEMORIA((ENTMEM1 * 255) + DESLOC1 + 33);
- PRIMENTIP ← MEMORIA((ENTMEM1 * 255) + DESLOC1 + 27);

3 - NUTUPS ← ⌊251/TAMTUP⌋ ÷ nº de tuplas por página
NUPAGS ← ⌈CARDCOR/NUTUPS⌉ ÷ nº de páginas lógicas ocupadas
NUTUPSULT ← CARDCOR - ((NUPAG - 1) * NUTUPS); ÷ nº de tuplas da última página ÷
POSREL ← INICIAL;
POSABS ← INICIAL + PRIMENTIE; ÷ entrada no INDESP
ACHADO ← FALSE;

4 - Para S ← Ø, incrementando de 1, até LIMATUAL OR ACHADO fazer

4.1 - chamar LETUPPOS (POSABS,1,PRIMENT,ENTMEM3,DESLOC3), a qual devolverá a tupla de INDESP que está na posição POSABS.PRIMENT é fixo e conhecido;

4.2 - TID ← MEMORIA ((ENTMEM3 * 255) + DESLOC3);
Se TID ≠ 0 então
fazer
-chamar LETUPTID (TID,PRIMENTIP,ENTMEM2,DESLOC2);
-Para I ← 0, incrementando de 1, até TAMCHAV - 1

```
                fazer
                  Se MEMORIA ((ENTMEM2 * 255) + DESLOC2 + I) ≠
                     MEMORIA ((ENTMEM  * 255) + DESLOC  + I)
                     então vá para 4.3;
                end;
                -ACHADO ← TRUE;
              end;
      4.3 -   POSREL ← (POSREL + SALTO) MOD (X * CARDMAX), onde
              X é um valor fixo e conhecido;
          -   POSABS ← PRIMENTIE + POSREL;
    end;

5 - Se ¬ ACHADO então ENTMEM2 ← "invalido";
```

B.8. LELIGDCHAV (ENTMEM1,DESLOC1,ENTMEM2,DESLOC2,POSRELLIGT, ENTMEM,DESLOC)

Objetivo - Ler uma tupla de uma tabela de ligados, de acordo com o valor da chave primária e posição relativa do ligante correspondente.

Parâmetros:

. ENTEMEM1,DESLOC1 - endereçam a tupla (na memória) (E) de DIRARQTAB que descreve a tabela de ligantes associada;

. ENTMEM2,DESLOC2 - endereçam a chave (na memória)
(E) da tupla que se quer ler;

. POSRELLIGT - informa a posição relativa do ligan-
(E) te associado;

. ENTMEM2,DESLOC2 - retorna o endereço (na memória)
(S) da tupla lida. Se ENTMEM2 = N ,
a tupla pretendida não se encon
tra na tabela.

Procedimento:

1 - Acessar a entrada em DIRARQTAB correspondente à tabela de ligados:

```
I ← POSDISP;
POSDISP ← POSDISP + 11;
MEMORIA ((N * 255) + I) ← "T";
```

-Para J ← 0, incrementando de 1, até 9
fazer

```
MEMORIA ((N * 255) + I + J + 1) ← MEMORIA ((ENTMEM1 * 255)
                                    + DESLOC1 + J);
```

end;

-chamar LETUPPOS (2, TAMANHO, PRIMENT, ENTMEM3, DESLOC3), a qual devolverá a tupla de DIRARQTAB que descreve o próprio DIRARQTAB.TAMANHO e PRIMEMT são fixas e conhecidas;

-chamar LETUPCHAV(ENTMEM3,DESLOC3,N,I,ENTMEM4,DESLOC4), a qual devolverá a tupla de DIRAQTRAB que descreve a tabela de ligados associada;

-POSDISP ← POSDISP - 11;

```
2 - TAMANHO ← MEMORIA ((ENTMEM1 * 255) + DESLOC1 + 26);
    PRIMENT1 ← MEMORIA ((ENTMEM1 * 255) + DESLOC1 + 27);

3 - Chamar LETUPPOS (POSRELLIGT,TAMANHO,PRIMENT1,ENTMEM3,
    DESLOC3), a qual devolverá a tupla de ligante correspondente;
    TID ← MEMORIA ((ENTMEM3 * 255) + DESLOC3 + TAMANHO - 2);
    TAMCHAV  ← MEMORIA ((ENTMEM4 * 255) + DESLOC4 + 32);
    DESLCHAV ← MEMORIA ((ENTMEM4 * 255) + DESLOC4 + 31);
    TAMANHO  ← MEMORIA ((ENTMEM4 * 255) + DESLOC4 + 26);
    PRIMENT1 ← MEMORIA ((ENTMEM4 * 255) + DESLOC4 + 27);
    ACHADO   ← False
    Enquanto ¬ACHADO
    fazer
      Se TID ≠ 0
      então  fazer
                -chamar LETUPTID (TID,PRIMENT1,ENTMEM,DESLOC) a
                 qual lerá o próximo ligado;
                -IGUAL ← True
                -Para J ← 0, incrementando de 1, até TAMCHAV or
                 ¬IGUAL
                 fazer
                   Se MEMORIA ((ENTMEM * 255) + DESLOC + DESLCHAV
                                                       + J) ≠
                      MEMORIA ((ENTMEM2* 255) + DESLOC2 + J)
                   então IGUAL ← False;
                 end;
```

```
                -Se IGUAL
                 então ACHADO ← True
                 senão TID ← MEMORIA ((ENTMEM * 255) + DESLOC +
                                                    TAMANHO - 2);
            end
      senão ENTMEM ← N;
end;
```

B.9. REMTUP (ENTMEM1, DESLOC1, ENTMEM2, DESLOC2, TID)

Objetivo - Remover uma tupla de uma tabela básica.

Parâmetros:

. ENTMEM1, DESLOC1 (E) - endereçam a tupla (na memória) de DIRARQTAB que descreve a tabela básica da qual a tupla será removida;

. ENTMEM2, DESLOC2 (E) - endereçam na memória a tupla a ser removida;

. TID (E) - informa o TID para a tupla a remover.

Procedimento:

1 - PRIMENTIP ← MEMORIA ((ENTMEM1 * 255) + DESLOC1 + 27);
PRIMENTIE ← MEMORIA ((ENTMEM1 * 255) + DESLOC1 + 28);
CARDCOR ← MEMORIA ((ENTMEM1 * 255) + DESLOC1 + 22);
CARDMAX ← MEMORIA ((ENTMEM1 * 255) + DESLOC1 + 24);
TAMTUP ← MEMORIA ((ENTMEM1 * 255) + DESLOC1 + 26);
TIPTAB ← MEMORIA ((ENTMEM1 * 255) + DESLOC1 + 11);

2 - Se PRIMEMTIE = 0 vá para 7;

3 - TAMCHAV ← MEMORIA ((ENTMEM1 * 255) + DESLOC1 + 32);
DESLCHAV← MEMORIA ((ENTMEM1 * 255) + DESLOC1 + 31);
DESLOC ← DESLOC2 + DESLCHAV;

4 - Chamar HASH(ENTMEM2,DESLOC,TAMCHAV,INICIAL,SALTO), a qual devolverá o "*home-address*" e o "salto" para a chave apresentada pela tupla a remover;
-POSREL ← INICIAL;
-LIMATUAL ← MEMORIA ((ENTMEM1 * 255) + DESLOC1 + 33);
-ACHADO ← False;

```
5 - Para I ← 0, incrementando de 1, até LIMATUAL or ACHADO
    fazer
      -POSABS ← POSREL + PRIMENTIE;
      -Chamar LETUPPOS (POSABS,2,PRIMENT,ENTMEM3,DESLOC3), a
       qual devolverá a tupla do Índice de Espalhamento cor-
       respondente a POSABS;
      -TID1 ← MEMORIA ((ENTMEM3 * 255) + DESLOC3);
      -Se TID1 = TID então ACHADO ← TRUE
                      senão POSREL ← (POSREL + SALTO) MOD (X * CARDMAX);
    end;

6 - MEMORIA (N * 255 + POSDISP) ← Ø;
    COPIAR  (N,POSDISP,ENTMEM3,DESLOC3,2);

7 - NUTUPS ← ⌊251/TAMTUP⌋
  - ULTPAGLOG ← ⌈CARDCOR/NUTUPS⌉
  - DESLULTTUP ← (CARDCOR - ((ULTPAGLOG - 1) * NUTUPS)) * TAMTUP+4;
  - Se TID(1) = ULTPAGLOG e TID(2) = DESLULTTUP
    então fazer
            PAGADESAL ← ENTMEM2;
            vá para 12;
          end;
  - chamar LETUPPOS (CARDCOR,TAMTUP,PRIMENTIP,ENTMEM4,DESLOC4),
    a qual devolverá a última tupla da tabela básica;
```

8 - Se PRIMENTIE = 0 vá para 11;
- chamar HASH (ENTMEM4,DESLOC4,DESLCHAV,TAMCHAV,INICIAL, SALTO) a qual devolverá o "*home-addres*" e o "salto" para a última tupla da tabela básica;
- POSREL ← INICIAL;
- ACHADO ← False;

9 - Para I ← 0, incrementando de 1, até LIMATUAL ou ACHADO fazer

POSABS ← POSREL + PRIMENTIE;
LETUPPOS (POSABS,2,PRIMENT,ENTMEM3,DESLOC3);
TID1 ← MEMORIA ((ENTMEM3 * 255) + DESLOC3);
Se TID1(1) = ULTPAGLOG e TID1(2) = DESLULTTUP
então ACHADO ← True
senão POSREL ← (POSREL + SALTO) MOD (X * CARDMAX);

end;

10 - MEMORIA (N * 255 + POSDISP) ← TID;
- COPIAR (N,POSDISP,ENTMEM3,DESLOC3,2), a qual fará com que o INDESP seja atualizado para apontar para a nova posição da tupla movida;

11 - COPIAR (ENTMEM4,DESLOC4,ENTMEM2,DESLOC2,TAMTUP) a qual copiará a última tupla da tabela básica no buraco deixado pela remoção;
- PAGADESAL ← ENTMEM4;

12 - Atualizar a cardinalidade:

-MEMÓRIA (N * 255 + POSDISP) ← CARDCOR - 1;

-COPIAR (N,POSDISP,ENTMEM1,DESLOC1 + 22,2);

13 - Se DESLULTTUP = 4

então - chamar DESALOCPAG (PAGADESAL), a qual devolverá a página física para a PILHADISP.

B.10. INCLIGD (ENTMEM1,DESLOC1,ENTMEM2,DESLOC2,POSREL,RESULT)

Objetivo - incluir uma tupla em uma tabela de ligados.

Parâmetros:

. ENTMEM1,DESLOC1 (E) - endereçam a tupla (na memória) de DIRARQTAB que descreve a tabela de ligantes associada a de ligados onde será feita a inclusão;

. ENTMEM2,DESLOC2 (E) - endereçam a tupla (na memória) a ser incluida;

. POSREL (E) - informa a posição relativa do ligante associado;

. RESULT (S) - retorna se a inclusão foi (0) ou não realizada.

Procedimento:

1 - I ← POSDISP;

- POSDISP ← POSDISP + 11;
- Para J ← 0, incrementando de 1, até 10

  fazer

    MEMORIA ((N * 255) + I + J + 1) ← MEMORIA ((ENTMEM1 * 255)
                                        + DESLOC1 + J);

  end;
- MEMORIA (N * 255 + I) ← "T";
- Chamar LETUPPOS (2,TAMTUP,PRIMENT1,ENTMEM3,DESLOC3), o qual devolverá a tupla de DIRARQTAB que descreve o próprio DIRARQTAB.TAMTUP e PRIMENT1 são fixas e conhecidas;
- chamar LETUPCHAV (ENTMEM3,DESLOC3,N,I,ENTMEM4,DESLOC4), a qual devolverá a tupla de DIRARQTAB que descreve a tabela básica onde será feita a inclusão;
- POSDISP ← POSDISP - 11;

2 - Ler a tupla do ligante associado:

- TAMTUP ← MEMORIA ((ENTMEM1 * 255) + DESLOC1 + 26);
- PRIMENT ← MEMORIA ((ENTMEM1 * 255) + DESLOC1 + 27);
- Chamar LETUPPOS (POSREL,TAMTUP,PRIMENT,ENTMEM3,DESLOC3);
- TID ← MEMORIA ((ENTMEM3 * 255) + DESLOC3 + TAMTUP - 2);

```
3 - CARDCOR ← MEMORIA ((ENTMEM4 * 255) + DESLOC4 + 22);
    TAMTUP  ← MEMORIA ((ENTMEM4 * 255) + DESLOC4 + 26);
    PRIMENT ← MEMORIA ((ENTMEM4 * 255) + DESLOC4 + 27);
    NUTUPS  ← ⌊251/TAMTUP⌋;
    NUPAGS  ← ⌈CARDCOR/NUTUPS⌉;
    Se (CARDCOR MOD NUTUPS) = 0
    então  fazer
             -NUPAGS ← NUPAGAS + 1;
             -Chamar ALOCPAGFIS (ENTMEM4,DESLOC4,NUPAGS,ENTMEM)
              a qual alocará uma nova página para a tabela  de
              ligados;
             -DESLOC ← 4;
           end
    senão  fazer
             -chamar LERPAGLOG (PRIMENT,NUPAGS,NUPAGFIS),
              o qual lerá a última página física da tabela  de
              ligados;
             -chamar OBTENTPAG (Ø, NUPAGFIS,PRIORI,ENTMEM);
             -DESLOC ← (CARDCOR - ((NUPAGS-1) * NUTUPS) + 1)
                                    * TAMTUP + 4;
           end;
    TAMCHAV  ← MEMORIA ((ENTMEM4 * 255) + DESLOC4 + 32);
    DESLCHAV ← MEMORIA ((ENTMEM4 * 255) + DESLOC4 + 31);
    RLINK  ← LLINK ← 0;
    ACHADO ← False;
```

```
4 - Enquanto ¬ACHADO
  - fazer
      Se TID ≠ Ø
      então fazer
              -chamar LETUPTID (TID,PRIMENT,ENTMEM1,DESLOC1),
               a qual lerá o próximo ligado;
              -MAIOR ← MENOR ← FALSE;
              -Para J ← 0, incrementando de 1, até TAMCHAV
                           ou MENOR ou MAIOR
               fazer
                 BYTE1 ← MEMORIA ((ENTMEM2 * 255) + DESLOC2 +
                                                  DESLCHAV + J);
                 BYTE2 ← MEMORIA ((ENTMEM1 * 255) + DESLOC1 +
                                                  DESLCHAV + J);
                 Se BYTE1 < BYTE2
                 então MENOR ← True
                 senão Se BYTE1 > BYTE2
                       então MAIOR ← True;
               end;
              -Se ¬MAIOR e ¬MENOR
               então  fazer
                        RESULT ← 1;
                        Se DESLOC = 4
                        então DESALOCPAG (ENTMEM);
                        vá para FIM;
                      end;
```

```
            -Se MAIOR
             então  fazer
                      LLINK ← TID;
                      TID ← MEMORIA ((ENTMEM1 + 255) +
                                          DESLOC1 + TAMTUP - 2);
                      RLINK ← TID;
                    end
             senão  fazer
                      RLINK ← TID
                      LLINK ← MEMORIA ((ENTMEM1 * 255) +
                                          DESLOC1 + TAMPTUP - 4);
                      ACHADO ← True
                    end;
        senão ACHADO ← True;
    end;
```

5 - Chamar COPIAR (ENTMEM2,DESLOC2,ENTMEM,DESLOC,TAMTUP), o qual copiará a tupla a incluir na última página lógica, após a última tupla presente;

-MEMORIA (N * 255 + POSDISP) ← (LLINK, RLINK);

-COPIAR (N, POSDISP, ENTMEM, DESLOC + TAMTUP - 4,4), a qual fará com que a nova tupla incluida aponte para a antecessora e sucessora;

-MEMORIA (N * 255 + POSDISP) ← (NUPAGS, DESLOC);

6 - Se LLINK ≠ 0

então fazer

-chamar GETUPTID (LLINK,PRIMENT,ENTEMEM3,DESLOC3), a qual lerá a tupla que precede a recém inserida, de acordo com o valor da chave;

-COPIAR (N,POSDISP,ENTMEM3,DESLOC3 + TAMTUP -2,2), a qual fará com que a tupla precedente aponte para a recém-incluida;

end

senão chamar COPIAR (N, POSDISP, ENTMEM3 , DESLOC3 + TAMANHO-2; 2), a qual faz com que o ligante aponte para a tupla incluida;

7 - Se RLINK ≠ 0

então COPIAR (N, POSDISP, ENTMEM4, DESLOC4 + TAMTUP - 4,2) qual faz com que a tupla sucessora na ordenação aponte para a recém-incluida;

8 - Atualizar a cardinalidade corrente:

-MEMORIA (N * 255 + POSDISP) ← CARDCOR + 1;

-COPIAR (N, POSDISP, ENTMEM4, DESLOC4 + 22,2).

B.11. REMLIGD (ENTMEM1, DESLOC1, ENTMEM2, DESLOC2, TID, POSREL)

Objetivo - remover uma tupla de uma tabela de ligados.

Parâmetros:

. ENTMEM1, DESLOC1 (E) - endereçam a tupla (na memória) de DIRARQTAB que descreve a tabela da qual a tupla será removida;

. ENTMEM2, DESLOC2 (E) - endereçam a tupla (na memória) a ser removida;

. TID (E) - informa o TID da tupla a remover;

. POSREL (E) - informa a posição relativa do ligante ao qual corresponde a tupla a ser removida.

Procedimento:

```
1 - TAMTUP  ← MEMORIA ((ENTMEM1 * 255) + DESLOC1 + 26);
    PRIMENT ← MEMORIA ((ENTMEM1 * 255) + DESLOC1 + 27);
    CARDCOR ← MEMORIA ((ENTMEM1 * 255) + DESLOC1 + 22);

2 - LLINK ← MEMORIA ((ENTMEM2 * 255) + DESLOC2 + TAMTUP - 4);
    RLINK ← MEMORIA ((ENTMEM2 * 255) + DESLOC2 + TAMTUP - 2);
```

```
3 - Se LLINK = 0
    então  fazer
             -Para I ← 0, incrementando de 1, até 9
              fazer
                MEMORIA((N * 255) + POSDISP + I) ←
                  ← MEMORIA((ENTMEM1 * 255) + DESLOC1 + I + 1);
              end;
              I ← POSDISP;
              POSDISP ← POSDISP + 10;
              -Chamar  LETUPPOS (2,TAMANHO, PRIMENT1, ENTMEM,
               DESLOC), a qual devolverá a tupla de DIRARQTAB
               que descreve o próprio DIRARQTAB.TAMANHO     e
               PRIMENT1 são fixos e conhecidos;
              -Chamar LETUPCHAV  (ENTMEM, DESLOC, N,I,ENTMEM3,
               DESLOC3), a qual devolverá a tupla de DIRARQTAB
               que descreve a tabela de ligantes associada;
              -POSDISP ← POSDISP - 10;
              -TAMANHO ← MEMORIA ((ENTMEM3 * 255) + DESLOC3 + 26);
              -PRIMENT1 ← MEMORIA ((ENTMEM3 * 255) + DESLOC3 + 27);
              -chamar LETUPPOS  (POSREL,TAMANHO,PRIMENT1,
               ENTMEM3,DESLOC3), a qual devolverá a tupla de li
               gantes correspondente;
              -MEMORIA (N * 255 + POSDISP) ← RLINK;
              -COPIAR (N,POSDISP,ENTMEM3,DESLOC3 + TAMANHO - 2,2);
    senão     -fazer
                 chamar LETUPTID (LLINK,PRIMENT,ENTMEM3,DESLOC3);
                 COPIAR (ENTMEM2,DESLOC2+TAMTUP-2,ENTMEM3,
                                    DESLOC3 + TAMTUP - 2,2);
              end;
```

```
4 - Se RLINK ≠ vazio
    então  fazer
             -Chamar LETUPTID (RLINK,PRIMENT,ENTMEM4,DESLOC4);
             -COPIAR (ENTMEM2,DESLOC2 + TAMTUP - 4, ENTMEM4,
              DESLOC4 + TAMTUP - 4,2);
           end;

5 - NUTUPS ← ⌊251/TAMTUP⌋;
    NUPAGS ← ⌈CARDCOR/NUTUPS⌉;
   -DESLULTTUP ← (CARDCOR - ((NUPAGS-1)*NUTUPS)*TAMANHO + 4;
   -Se NUPAGS = TID(1) e DESLULTTUP = TID(2)
    então  fazer
             PAGADESAL ← ENTMEM2;
             vá para 6;
           end;
  -chamar  LETUPPOS (CARDCOR,TAMTUP,PRIMENT,ENTMEM3,DESLOC3), a
   qual devolverá a última tupla da tabela de ligados;
  -RLINK ← MEMORIA ((ENTMEM3 * 255) + DESLOC3 + TAMTUP - 2);
  -LLINK ← MEMORIA ((ENTMEM3 * 255) + DESLOC3 + TAMTUP - 4);
  -Se LLINK ≠ 0
   então fazer
           -chamar LETUPTID (LLINK,PRIMENT,ENTMEM4,DESLOC4);
           -MEMORIA (N * 255 + POSDISP) ← TID;
           -COPIAR (N,POSDISP,ENTMEM4,DESLOC4 + TAMTUP - 2,2);
         end;
```

-Se RLINK ≠ 0

então fazer

-chamar LETUPTID (RLINK,PRIMENT,ENTMEM4,DESLOC4);

-MEMORIA (N * 255 + POSDISP) ← TID;

-COPIAR (N,POSDISP,ENTMEM4,DESLOC4 + TAMTUP - 4,2);

6 - Atualizar a cardinalidade:

-MEMORIA (N * 255 + POSDISP) ← CARDCOR - 1;

-COPIAR (N,POSDISP,ENTMEM1,DESLOC1 + 22,2);

-PAGADESAL ← ENTMEM3;

7 - Se DESLULTTUP = 4

então - chamar DESALOCPAG (PAGADESAL), a qual devolverá a página para PILHADISP.

B.12. LIGAHASH (ENTMEM1,DESLOC1,ENTMEM2,DESLOC2,TID,RESULT)

Objetivo - Incluir a chave da tupla sendo incluida, no Índice de Espalhamento.

Parâmetros:

. ENTMEM1,DESLOC1 (E) - endereçam a tupla (na memória) de DIRARQTAB que descreve a tabela na qual a nova tupla está sendo incluida;

. ENTMEM2,DESLOC2 (E) - endereçam a tupla (na memória) que se quer incluir;

. TID (E) - informa o TID para a tupla a ser incluida; e

. RESULT (E) - retorna a resultado da operação, ou seja: 0(com sucesso) e 1(sem sucesso).

Procedimento:

```
1 - PRIMENTIP ← MEMORIA ((ENTMEM1 * 255) + DESLOC1 + 27);
  - PRIMENTIE ← MEMORIA ((ENTMEM1 * 255) + DESLOC1 + 29);
  - DESLCHAV  ← MEMORIA ((ENTMEM1 * 255) + DESLOC1 + 31);
  - TAMCHAV   ← MEMORIA ((ENTMEM1 * 255) + DESLOC1 + 32);
  - CARDMAX   ← MEMORIA ((ENTMEM1 * 255) + DESLOC1 + 24);
  - LIMATUAL  ← MEMORIA ((ENTMEM1 * 255) + DESLOC1 + 33);
  - LIMMAX    ← False;
```

- Chamar HASH (ENTMEM2,DESLOC2 + DESLCHAV, TAMCHAV, INICIAL, SALTO);
- SLIM ← LIMATUAL;
- Chamar LETUPPOS (3,2,PRIMENT,ENTMEM,DESLOC) a qual lerá a tupla de DIRARQTAB que descreve o INDESP.
  PRIMENT é fixo e conhecido.
- PRIMENT ← MEMORIA ((ENTMEM * 255) + DESLOC + 27);

2 - Enquanto ¬LIMMAX

fazer

HS ← INICIAL;

S ← 0;

VAGO ← TROCA ← False;

Enquanto S ≤ SLIM e ¬VAGO

fazer

-ENTRADAIE ← PRIMENTIE + HS;

-Chamar LETUPPOS (ENTRADAIE,2,PRIMENT,ENTMEM,DESLOC), a qual lerá a tupla de INDESP correpondente a HS;

-TID1 ← MEMORIA ((ENTMEM * 255) + DESLOC);

-Se TID1 = 0 então VAGO ← True

senão fazer

S ← S + 1;

HS ← (HS+SALTO) MOD (X*CARDMAX);

end;

end;

```
Se ¬VAGO
então fazer
        LIMI ← SLIM; LIM ← SLIM + 1;
        para I ← 0, incrementando de 1, enquanto I ≤ LIMI
        fazer
          ENDI ← (INICIAL + I * SALTO) MOD (X*CARDMAX);
          ENTRADAIE ← PRIMENTIE + ENDI;
          Chamar LETUPPOS (ENTRADAIE,2,PRIMENT,ENTMEM3,
                                DESLOC3)
          TID2 ← MEMORIA ((ENTMEM3 * 255) + DESLOC3);
          LETUPTID (TID,PRIMENTIP,ENTMEM4,DESLOC4);
          HASH (ENTMEM4,DESLOC4 + DESLCHAV,TAMCHAV,
                INICIAL1,SALTO1);
          HS2 ← (INICIAL1 + SALTO1) MOD (X * CARDMAX);
          ENTRADAIE ← PRIMENTIE + HS2;
          LETUPPOS (ENTRADAIE,2,PRIMENT,ENTMEM4,DESLOC4);
          TID3 ← MEMORIA ((ENTMEM4 * 255) + DESLOC4);
          S2 ← 1;
          Enquanto S2 < LIM e TID3 ≠ 0
          fazer
            S2 ← S2 + 1;
            HS2 ← (HS2 + SALTO1) MOD (X * CARDMAX);
            ENTRADAIE ← PRIMENTIE + HS2;
            LETUPPOS (ENTRADAIE,2,PRIMENT,ENTMEM4,
                          DESLOC4);
            TID3 ← MEMORIA ((ENTMEM4 * 255) + DESLOC4);
          end;
```

```
            Se S2 < LIM então
            fazer
              TROCA ← True
              LIMI  ← -1;
            end;
          end;
          Se TROCA
          então  fazer
                   MEMORIA (N* 255 + POSDISP) ← TID;
                   COPIAR (ENTMEM3,DESLOC3,ENTMEM4,DESLOC4,2);
                   COPIAR (N,POSDISP,ENTMEM3,DESLOC3,2);
                   Se SLIM ≠ LIMATUAL então
                   fazer
                     MEMORIA (N * 255 + POSDISP) ← SLIM;
                     COPIAR  (N,POSDISP,ENTMEM1,DESLOC1+33,1);
                   end
                   LIMMAX ← True;
                 end
          senão  Se SLIM < "limite-máximo"
                 então SLIM ← SLIM + 1
                 senão  fazer
                          LIMMAX ← True;
                          RESULT ← 1;
                        end;
end
```

```
        senão fazer
                MEMORIA (N * 255 + POSDISP) ← TID;
                COPIAR  (N,POSDISP,ENTMEM,DESLOC,2);
                LIMMAX ← True;
              end;
    end;
```

B.13. CRIATAB (ENTMEM,DESLOC)

Objetivo - Criar uma tabela básica.

Parâmetros:

. ENTMEM,DESLOC (E) - endereçam a tupla (na memória) de DIRARQTAB que descreve a tabela a ser criada.

. RESULT (S) - retorna se a operação foi (0) ou não (1) realizada.

Procedimento:

```
1 - CARDMAX ← MEMORIA ((ENTMEM * 255) + DESLOC + 24);
    TAMTUP  ← MEMORIA ((ENTMEM * 255) + DESLOC + 26);
    NUTUPS  ← ⌊251/TAMTUP⌋;
    NUPAGS  ← ⌈CARDMAX/NUTUPS⌉;
```

2 - Chamar LETUPPOS (1 ,TAMANHO,PRIMENT,ENTMEM1,DESLOC1), a qual lerá a tupla de DIRARQTAB que descreve o INDPAG.TAMANHO e PRIMENT são fixos e conhecidos;

3 - CARDCOR ← MEMORIA ((ENTMEM1 * 255) + DESLOC1 + 22);
CARDMAX1 ← MEMORIA ((ENTMEM1 * 255) + DESLOC1 + 24);
CARDCOR ← CARDCOR + NUPAGS;
Se CARDCOR > CARDMAX então fazer
RESULT ← 1;
vá para FIM;
end

4 - PRIMENTIE ← MEMORIA ((ENTMEM * 255) + DESLOC + 29);
Se PRIMENTIE = 0 então vá para 8;

5 - Chamar LETUPPOS (3,TAMANHO,PRIMENT,ENTMEM2,DESLOC2), o qual lerá a tupla de DIRARQTAB que descreve o INDESP. TAMANHO e PRIMENT são fixas e conhecidas.

6 - CARDCOR1 ← MEMORIA ((ENTMEM2 * 255) + DESLOC2 + 22);
CARDMAX2 ← MEMORIA ((ENTMEM2 * 255) + DESLOC2 + 24);
CARDCOR1 ← CARDCOR1 + (X * CARDMAX);
Se CARDCOR1 < CARDMAX2
então fazer
RESULT ← 2;
vá para FIM;
end;

7 - MEMORIA (N * 255 + POSDISP) ← CARDCOR1;
- COPIAR (N,POSDISP,ENTMEM2,DESLOC2 + 22,2), a qual atualizará a cardinalidade do INDESP;
- MEMORIA (N * 255 + POSDISP) ← CARDCOR1 - (X * CARDMAX) + 1;
- COPIAR (N,POSDISP,ENTMEM,DESLOC + 29,2), a qual preencherá o campo "1ª entrada em INDESP" para a tabela sendo criada,

8 - MEMORIA (N * 255 + POSDISP) ← CARDCOR;
- COPIAR (N,POSDISP,ENTMEM1,DESLOC1 + 22,2), a qual atualizará a cardinalidade corrente do INDPAG;
- MEMORIA (N * 255 + POSDISP) ← CARDCOR - NUPAGS + 1;
- COPIAR (N,POSDISP,ENTMEM,DESLOC + 27,2), a qual preencherá o campo "1ª entrada em INDPAG" para a tabela sendo criada.

B.14. REMOVETAB (ENTMEM,DESLOC)

Objetivo - Remover uma tabela básica.

Parâmetros:

. ENTMEM,DESLOC (E) - endereçam a tupla (na memória) de DIRARQTAB que descreve a tabela a ser removida.

Procedimento:

1 - CARDCOR ← MEMORIA ((ENTMEM * 255) + DESLOC + 22);
CARDMAX ← MEMORIA ((ENTMEM * 255) + DESLOC + 24);
TAMTUP ← MEMORIA ((ENTMEM * 255) + DESLOC + 26);
PRIMENTIP ← MEMORIA ((ENTMEM * 255) + DESLOC + 27);
PRIMENTIE ← MEMORIA ((ENTMEM * 255) + DESLOC + 29);
NUTUPS ← ⌊251/TAMTUP⌋;
NUPAGSOCU ← ⌈CARDCOR/TAMTUP⌉;
NUPAGSALO ← ⌈CARDMAX/TAMTUP⌉;

2 - Chamar LETUPPOS (1,TAMANHO,PRIMENT,ENTMEM1,DESLOC1), a qual devolverá a tupla de DIRARQTAB que descreve o INDAPAG. TAMANHO e PRIMENT são fixos e conhecidos.
- CARDCOR1 ← MEMORIA ((ENTMEM1 * 255) + DESLOC1 + 22);
- LIMITESUP ← CARDCOR1 - NUPAGSALO - PRIMENTIP;
- TUP1 ← PRIMENTIP
- TUP2 ← PRIMENTIP + NUPAGSALO;

3 - Para I ← 0, incrementando de 1, até LIMITESUP
fazer
-chamar LETUPROS (TUP1,2,1,ENTMEM2,DESLOC2);
-Se I < NUPAGSOCU
então fazer
PAGFIS ← MEMORIA ((ENTMEM2 * 255) + DESLOC2);
chamar OBTENPAG (0,PAGFIS,PRIORI,ENTMEM3);
chamar DESALOCPAG (ENTMEM3);
end;

-chamar LETUPPOS (TUP2,2,1,ENTMEM3,DESLOC3);

-chamar COPIAR (ENTMEM3,DESLOC3,ENTMEM2,DESLOC2,2);

-TUP1 ← TUP1 + 1;

-TUP2 ← TUP2 + 1;

end;

4 - MEMORIA (N * 255 + POSDISP) ← CARDCOR1 - NUPAGSALO;

- COPIAR (N,POSDISP,ENTMEM1,DESLOC1 + 22,2), a qual atualizará a cardinalidade do Índice de Páginas;

5 - Se PRIMENTIE ≠ 0 então

fazer

-chamar LETUPPOS (3,TAMANHO,PRIMENT,ENTMEM1,DESLOC1), a qual lerá a tupla de DIRARQTAB que descreve o INDESP. TAMANHO e PRIMENT são fixos e conhecido;

-CARDCOR1 ← MEMORIA ((ENTMEM1 * 255) + DESLOC1 + 22);

-TUP1← PRIMENTIE;

-TUP2← PRIMENTIE + (X * CARDMAX);

-LIMITESUP ← CARDCOR1 - (X * CARDMAX) - PRIMENTIE;

-Para I ← 0, incrementando de 1, até LIMITESUP

fazer

chamar LETUPPOS (TUP1,2,PRIMEMT1,ENTMEM2,DESLOC2);

chamar LETUPPOS (TUP2,2,PRIMENT1,ENTMEN3,DESLOC3);

COPIAR (ENTMEM3,DESLOC3,ENTMEM2,DESLOC2,2);

end;

-MEMORIA (N * 255 + POSDISP) ← CARDCOR1 - (X * CARDMAX);

-COPIAR (N,POSDISP,ENTMEM1,DESLOC1,2), a qual atualizará a cardinalidade de INDESP;

6 - chamar LETUPPOS (2,TAMANHO,PRIMENT,ENTMEM1,DESLOC1), a qual devolverá a tupla de DIRARQTAB que descreve o próprio DIRARQTAB;

- CARDCOR1 ← MEMORIA ((ENTMEM1 * 255) + DESLOC1 + 22);
- Para I ← 1, incrementando de 1, até CARDCOR1

fazer

-chamar LETUPPOS (I,TAMANHO,PRIMENT,ENTMEM2,DESLOC2), a qual lerá uma tupla de DIRARQTAB;

-PRIMENTIP1 ← MEMORIA ((ENTMEM2 * 255) + DESLOC2 + 27);

-PRIMENTIE1 ← MEMORIA ((ENTMEM2 * 255) + DESLOC2 + 29);

-Se PRIMENTIP1 > PRIMENTIP então

fazer

-MEMORIA (N * 255 + POSDISP) ← PRIMENTIP1 - NUPAGSALO;

-COPIAR (N,POSDISP,ENTMEM2,DESLOC2 + 27,2), a qual atualizará o campo "1ª entrada em INDPAG";

end;

-Se PRIMENTIE1 > PRIMENTIE então

fazer

MEMORIA (N * 255 + POSDISP) ← PRIMENTIE1 - (X * CARDMAX);

COPIAR (N, POSDISP, ENTMEM2, DESLOC2 + 29,2), a qual atualizará o campo "1ª entrada em INDESP".

end;

C.1. RECSEC (NUSEC, UDISCO)

Objetivo - Promover a "reconstrução de seções".

Parâmetros:

. NUSEC - informa o "nº da seção" até a qual se quer (E) que a Base de Dados seja reconstruida;

. UDISCO - informa a unidade de disco na qual será (E) feita a reconstrução.

Procedimento:

1 - Chamar LEREGSOM (DISCREC,1,N) a qual lerá o registro 1 do disquete de reconstrução (DISCREC) para a página N (de trabalho) da memória de páginas;

```
2 - NUPRIMSEC ← MEMORIA ((N * 255) + 4);
    INDSEC ← NUSEC - NUPRIMSEC + 2;
    PAGINI ← MEMORIA ((N * 255) + 4 + (INDSEC  * 2));
    K ← -1;
    REGCORRENTE ← MEMORIA (N * 255);
    Se UDISCO ≠  DISCBD então
    fazer
      Para I ← 0, incrementando de 1, até 1024
      fazer
        chamar LEREGSOM (DISCBD,I,0);
        chamar GRVREGSOM (UDISCO,0,I);
      end;
    end;
```

```
3 - Para I ← PAGINI, incrementando de 1 até REGCORRENTE
    fazer
      chamar LEREGSOM (DISCREC,I,Ø);
      NUPAG ← MEMORIA (Ø);
      Para J ← 0, incrementando de 1, até K
      fazer
        Se MEMORIA (255 + (2 * J)) = NUPAG
        então vá para FIMLACO;
      end;
      K ← K + 1;
      MEMORIA (255 + (2 * K)) ← NUPAG;
      Chamar GRVREGSOM (UDISCO, Ø, NUPAG);
      FIMLACO:
    end;

4 - Se UDISCO ≠ DISCBD
    então fazer
            -MEMORIA (N * 255) ← 1;
            -MEMORIA (N * 255 + 2) ← 0;
            -MEMORIA (N * 255 + 4) ← 0;
            -Pedir para substituir o disquete que está em
             DISCREC pelo disquete que conterá a fita-log do
             novo ACSET criado;
          end
    senão fazer
             MEMORIA (N * 255) ← PAGINI - 1;
             MEMORIA (N * 255 + 2) ← NUSEC - NUPRIMSEC + 1;
          end;
```

5 - Chamar GRVREGSOM (DISCREC, N, 1) a qual gravará as informações sobre a fita-log no registro 1 do disquete correspondente.

C.2. RECCOM (NUCOMANDO)

Objetivo - Reconstruir a base de dados até o estado em que ela se encontrava após a execução do comando NUCOMANDO.

Parâmetros:

. NUCOMANDO - informa até qual comando se quer que (E) a BD seja reconstruida.

Procedimento:

1 - Chamar LIMPAMEM, a qual limpará a memória de páginas;

2 - Chamar LEREGSOM (DISCREC, 1, N), a qual lerá o 1º registro do disquete de reconstrução (DISCREC) para a página N;

3 - NUSECOES ← MEMORIA ((N * 255) + 2);
PAGINI ← MEMORIA ((N * 255) + 4 + (NUSECOES * 2));
REGCORRENTE ← MEMORIA (N * 255);

```
4 - Para I ← PAGINI, incrementando de 1, até REGCORRENTE
    fazer
      chamar LEREGSOM  (DISCREC,I,0);
      NUCOM ← MEMORIA(2);
      Se NUCOM ≥ NUCOMANDO + 1 então vá para 5;
    end;
    vá para FIM;

5 - K ← 0;
    NUPAG ← MEMORIA (0);
    Chamar GRVREGSOM (DISCBD, 0, NUPAG);
    MEMORIA (255) ← NUPAG;

6 - Para J ← I + 1, incrementando de 1, até REGCORRENTE
    fazer
      chamar LEREGSOM  (DISCREC, J, 0);
      NUPAG ← MEMORIA (0);
      Para L ← 0, incrementando de 1, até K
      fazer
        Se MEMORIA (255 + (L * 2)) = NUPAG então vá para FIMLACO;
      end;
      K ← K + 1;
      MEMORIA (255 + (K * 2)) ← NUPAG;
      chamar GRVREGSOM (DISCBD, 0, NUPAG);
      FIMLACO:
    end;
```

7 - MEMORIA (N * 255) ← I - 1;
-chamar GRVREGSOM (DISCREC, N, 1), a qual gravará as informações atualizadas sobre a fita-log.

C.3. LIMPAFITA

Objetivo - retirar da fita-log as cópias intermediárias das páginas relativas à seção corrente.

Procedimento:

1 - Chamar LEREGSOM (DISCREC, 1, N), a qual lerá as informações sobre a fita-log para a página N;

2 - REGCORRENTE ← MEMORIA (N * 255);
NUSECREGS ← MEMORIA ((N * 255) + 2);
PAGINI ← MEMORIA ((N * 255) + 4 + (NUSECREGS * 2));
K ← -1;

3 - Para I ← PAGINI, incrementado de 1, até REGCORRENTE
fazer
Chamar LEREGSOM (DISCREC, I, 0);
NUPAG ← MEMORIA (0);
Para J ← 0, incrementando de 1, até K
fazer
Se MEMORIA (255 + (2 * J)) = NUPAG
então vá para FIMLACO;
end;

```
      K ← K + 1;
      Se I ≠ PAGINI + K então
      fazer
        chamar GRVREGSOM (DISCREC, 0, PAGINI + K);
      end;
      MEMORIA (255 + (2 * K)) ← NUPAG ;
      FIMLACO:
    end;

4 - MEMORIA (N * 255) ← PAGINI + K;
    chamar GRVREGSOM (DISCREC, N, 1);
```

BIBLIOGRAFIA

|1| D'Albuquerque, V.L., "Gramática da Linguagem LOBAN 1980", Relatório Técnico, COPPE/UFRJ, 1981.

|2| D'Albuquerque, V.L., Tese de M.Sc., COPPE/UFRJ, em preparação.

|3| Dantas, J.S., "Modelagem de um Sistema Integrado de Informação usando LOBAN", Relatório Técnico, COPPE/UFRJ, 1982.

|4| Durchholz, R., Richter, G., "Concepts for Data Management Systems", North. Holland Pub. Co., Amsterdam, 1974.

|5| Durchholz, R., Richter, G., "Information Management Concepts for use with DBMS Interfaces", North Holland Pub. Co., Amsterdam, 1976.

|6| Heuser, A.C. et alli, "Sistema L: Uma Implementação da Linguagem LOBAN", Anais do VIII SEMISH, 1981.

|7| Pinto, P.R.B., "Aspectos Conceituais sobre Concorrência em Bando de Dados", Tese de M.Sc., COPPE/UFRJ, 1979.

|8| Richter, G., Pereira Filho, J.C., Castilho, J.M.V., "Projeto MINIBAN - Relatório Final da Segunda Etapa - Parte A. MINIBAN 29", Relatório Técnico, 1978.

|9| Richter, G., Castilho, J.M.V., "Uma Interface para Sistemas de Informação: LOBAN - Linguagem de Operação de Banco de Dados", Anais do 11º CNPD - SUCESU, 1978.

|10| Richter, G., "On the Relationship Between Information and Data", Lectures Notes in Computer Science, vol. 39, Springer-Verlag, Berlim, 1976.

|11| Santos, A.C., "Estruturas de Dados para a Interface de Banco de Dados LOBAN", Tese de M.Sc., COPPE/UFRJ, 1981.

|12| Santos, A.C. et alli, "Especificação da Linguagem LOBAN, 1980", Versão 2, Relatório Técnico, COPPE/UFRJ, 1981.

|13| Simone, E.G., Teles, A.A., "Gerador de Analisadores Sintáticos RRP LL(1)", Anais VIII SEMISH, 1981.

|14| Sousa, A.C.G. et alli, "O Banco de Dados MICROLOBAN", Anais do IX SEMISH, 1982.

|15| Souza, J.M., "Algoritmos de Hashing para Problemas Específicos", Tese de M.Sc., COPPE/UFRJ, 1978.

|16| Souza, J.M., "Uma Experiência com Implementação de Sistemas de Banco de Dados Didáticos", RBC. Vol. 1, Nº 2, 1981.

|17| Souza, J.M. et alli, "COPPEREL: Sistema de Gerência de Base de Dados da COPPE", Relatório Técnico, COPPE/UFRJ,1982.

|18| Astrahan, M.M. et alli, "Sistema R: A Relational Approach to Database Management", ACM-TODS, Vol. 1, Nº 2, 1967.

|19| Blasgen, M.W., Eswaran, K.P., "Storage and Access in Relational Data Bases", IBM Syst. J, Nº 4, 1977.

|20| Brent, R.P., "Reducing the Retrieval Time of Scatter Storage Techniques", CACM, Vol. 16, Nº 2, 1973.

|21| Clapson, P., "Improving the Access Time for Randon Access Files", CACM, Vol. 20, Nº 3, 1977.

|22| Date, C.J., "An Introduction to Database Systems", Third Edition, Addison-Wesley, 1981.

|23| Knuth, D., "The Art of Computer Programming, Vol. 3: Sorting and Searching", Reading-Mass, Addison-Wesley, 1973.

|24| Lorie, R.A., Nilsson, J.F., "An Access Specification Language for a Relational Data Base System", IBM J. RES. DEVELOP, Vol. 23, Nº 3, 1979.

|25| Lum, V.Y., Yuen, P.S.T., Dodd, M., "Key-to-address transform techniques. A Fundamental Performance Study on Large Existing Formatted Files", CACM, Vol. 14, Nº 4, 1971.

|26| Mylopoulos, J. et alli, "A Multi-level Relational System", Proc. NCC, AFIPS Press, 1975.

|27| Severance, D.G., Carlis, J.V., "A Pratical Approach to Selecting Record Access Paths", ACM-Computing Surveys, Vol. 9, Nº 4, 1977.

|28| Stonebreaker, M., Wong, E., Kreps, P., "The Design and Implementation of INGRES", ACM-TODS, Vol. 1, Nº 3, 1976.

|29| Tsichiritzis, D., "LSL: A Link and Selector Language", Relatório Técnico, University of Toronto, 1976.

|30| Verhofstad, J.S.M., "Recovery Techniques for Database Systems", ACM-Computing Surveys, Vol. 10, Nº 2, 1978.

[31] COBRA, "Manual de Referência do Sistema Operacional Monoprogramável - SOM COBRA/TD", T3V0100002-0, 1980.

[32] ANSI/X3/SPARC Study Group on Data Base Management Systems: Interim Report. FDT (Bulletin of ACM SIGMOD), Vol. 2, Nº 2, 1975.