IMPLEMENTAÇÃO DE UM SISTEMA DE GERENCIAMENTO DE BANCO DE DADOS RELACIONAL

MARIA DE FÁTIMA R.O.P. DA SILVA

UNICAMP

UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE MATEMÁTICA, ESTATÍSTICA E CIÊNCIA DA COMPUTAÇÃO

CAMPINAS - SÃO PAULO
BRASIL

IMPLEMENTAÇÃO DE UM SISTEMA DE GERENCIAMENTO DE BANCO DE DADOS RELACIONAL

MARIA DE FÁTIMA R. O. P. DA SILVA

Orientador

Prof.Dr. Nelson Castro Machado

Dissertação apresentada ao Instituto de Matemática, Estatística e Ciência da Computação como requisito parcial para obtenção do título de Mestre em Ciência da Computação.

setembro - 1984

Este exemplar corresponde a redação final da tese defendida pela Senhora Maria de Fátima R. O. P. da Silva e aprovada pela Comissão Julgadora.

Campinas, 24 de Outubro de 1984

RESUMO

Devido ao crescente afluxo de informações que devem estar disponíveis a um número de pessoas cada vez maior em um espaço de tempo menor, a área de bancos de dados vem merecendo nos últimos anos uma atenção especial. Sistemas de gerenciamento de bancos de dados têm como objetivo principal gerenciar, de uma forma eficiente e integrada, o armazenamento e acesso a essas informações.

Esse trabalho descreve a implementação do BIBLOS, um sistema de gerenciamento de banco de dados relacional. Inicialmente é descrito o histórico do projeto e em seguida são dados alguns conceitos fundamentais da área de banco de dados, para que o leitor se familiarize com a terminologia usada. Finalmente o projeto é apresentado através de sua estrutura de dados, métodos de armazenamento e acesso, módulos de manutenção e recuperação. Segue-se a descrição da implementação da linguagem de recuperação utilizada: SEQUEL 2.

A experiência obtida demonstrou que, para que um DBMS atenda a seus objetivos é fundamental a utilização de técnicas adequadas e a atenção a inúmeros detalhes práticos. Para tornar o BIBLOS mais eficiente recomenda-se nas conclusões a incorporaçao de algumas técnicas sofisticadas de armazenamento e acesso.

ABSTRACT

Due to the ever increasing amount of information that must be made available to a large number of persons in the least possible time, the Data Base area has been extremely active in the last few years. Data Base Management Systems are essentially responsible for the efficient and integrated management of the storage and access to such information.

This paper describes the implementation of BIBLOS, a relational Data Base Management System. The historic of the project is iniatially described,followed by a few fundamental concepts in Data Bases for the benefit of readers not familiar with the area. Then the project itself is presented, including the data structure, storage and access methods and maintenance and retrieval modules. The implementation of the query language used: SEQUEL 2 is also described.

The experience provided by this effort has shown that, in order to obtain the basic objectives of a DBMS, it is fundamental to utilize adequate implementation techniques and to give attention to several practical considerations. In order to make BIBLOS more efficient, the introduction of a few sophisticated storage and access techniques is suggested.

SUMÁRIO

1 - INTRODUÇÃO

1.1 - BANCOS DE DADOS NA ATUALIDADE

Nos últimos dez anos, uma área da computação vem se desenvolvendo com muita rapidez e com a adição frequente de técnicas mais eficientes e sofisticadas: Trata-se da área de bancos de dados que juntamente com a área de teleprocessamento e redes, vem merecendo na literatura um espaço cada vez maior.

Isto se deve em grande parte ao crescente afluxo de informações que devem estar disponíveis a um maior número de pessoas no menor espaço de tempo. Torna-se pois necessário desenvolver tecnologia que gerencie o armazenamento, a integração e o acesso a essas informações para que se possa usar recursos sempre limitados para atender a demanda essencialmente ilimitada.

Daí o interesse crescente em sistemas de gerenciamento de bancos de dados.

1.2 - HISTÓRICO DO DESENVOLVIMENTO DO PROJETO

O objetivo inicial do projeto era únicamente o desenvolvimento de técnicas simples e eficientes para a criação de um sistema de recuperação de informações que atendesse às necessidades da instituição (UNICAMP) e fôsse compatível com seus recursos computacionais relativamente limitados.

Assim, foi desenvolvida e implementada uma primeira versão do sistema, bastante simplificada, que visava essencialmente atender às necessidades da Biblioteca Central da UNICAMP, conforme descrito em 3.1.

O sistema foi demonstrado e correspondeu / plenamente à expectativa, muito embora não tenha sido totalmente/ implantado devido à falta de recursos na unidade interessada. Entrementes, outras aplicações foram surgindo forçando ampliações na capacidade do sistema e aperfeiçoamentos nas técnicas utilizadas no projeto inicial.

Exemplificando, foi necessária a diversificação do tipo de informações armazenáveis, inicialmente constituídas apenas por caracteres ASCII, para incluir dados em forma binária. A diversificação das aplicações mostrou a necessidade de tornar o sistema capaz de manipular uma ampla variedade de tipos de dados, levando à criação de um dicionário de dados. Este dicionário descreve as características de uma aplicação tais como número de registros atualmente disponíveis, número de campos existentes,

nomes desses campos, tipos e formatos dos dados, etc..., efetivamente configurando o sistema para cada aplicação particular.

Nova e importante extensão foi exigida por certas aplicações que requeriam mais de um arquivo principal de armazenamento dos dados, devido à existência de mais de um tipo / de informação básica (ex: arquivo de professores, arquivo de alunos, arquivo de disciplinas, etc...). Estas informações deveriam/ ser gerenciadas de maneira a permitir manipulação simultânea, incluindo a utilização de informação obtida em um arquivo para ter acesso a dados localizados em outro. Isto é, foi necessário adicionar ao sistema a capacidade de acesso relacional.

Em qualquer sistema de recuperação de informações é indispensável implementar uma interface de comunicação entre usuário e sistema. Esta interface fornece ao usuário / uma linguagem com recursos para consulta, manipulação e criação / de informações, permitindo ainda, no caso mais geral, a obtenção/ de relatórios contendo dados dos diversos arquivos em formatos especificados pelo usuário.

À medida que o sistema evoluia, a interface com o usuário, inicialmente limitada a uma simples linguagem / de consulta voltada para aplicação específica (recuperação de referencias bibliográficas), teve que ser consideràvelmente ampliada, levando à implementação de grande parte de uma sofisticada / linguagem de consulta de propósito geral.

Finalmente, para que os dados possam ser utilizados eficientemente, deve ser permitido a vários usuários / acesso simultâneo ao sistema. Para tanto seria necessário criar

todo um esquema de proteção que garantisse a privacidade e integridade dos dados em consultas e atualizações simultâneas. Assim sendo, levou-se também em conta esta área no desenvolvimento do sistema, muito embora o trabalho tenha se concentrado apenas no aspecto de integridade.

Em suma, o projeto que no início visava / simplesmente a exploração de técnicas eficientes para o armazenamento e recuperação de informações, evoluiu para a implementação / de um verdadeiro sistema de gerenciamento de banco de dados relacional, com ênfase na parte de recuperação de informações, que recebeu o nome de BIBLOS devido à primeira aplicação originalmente/ contemplada.

O capítulo seguinte é dedicado a uma introdução sumária dos conceitos fundamentais da área de banco de dados, para benefício do leitor não familiarizado. Os demais capítulos descrevem o sistema BIBLOS.

2 - CONCEITOS FUNDAMENTAIS DA ÁREA DE BANCOS DE DADOS

2.1 - INTRODUÇÃO

Entende-se por Sistema de Gerenciamento de Banco de Dados (daqui em diante denotado pela sigla tradicional / DBMS, de suas iniciais em inglês) a um recurso de caráter geral para a manipulação eficiente e centralizada de grandes coleções / de informações estruturadas e armazenadas de uma forma consistente e integrada. Tais sistemas (DBMS's) são constituídos por software dedicado para consulta, manutenção e análise dos dados incluindo em especial as interfaces entre o sistema e o usuário, / sob a forma de uma ou mais linguagens que possibilitem o acesso aos dados, de forma conveniente, a uma ampla gama de usuários que pode incluir desde leigos até programadores experientes.

As origens dos DBMS's têm sido identificadas desde fins da década de 50, quando autores como Mc Gee [1,2] discutiram o sucesso do emprego de rotinas "generalizadas". Propunha-se, por exemplo, ao invés de escrever um programa específico/ para cada tarefa de classificação de arquivos, o uso de rotinas / capazes de classificar "qualquer" arquivo independentemente dos dados em particular que contivesse (o usuário apenas fornece os parâmetros para dirigir os principais elementos do processo de classificação); estes autores propuseram que essas idéias fossem estendidas a outras áreas de processamento de dados tais como ma-

nutenção de arquivos e geração de relatórios. Isso implica na construção de funções especiais que executam tarefas de processamento de dados mais comuns e frequentemente usadas.

Mas tal generalidade não é obtida sem custo: o preço do processamento generalizado é a redução da eficiência da operação, geralmente devido ao processamento interpretativo ou uso mais intenso de recursos de hardware do sistema, tais / como a capacidade de memória rápida, memória de massa, tempo de CPU, etc...

O desenvolvimento do hardware nas últimas/ duas décadas provocou uma diminuição significativa na razão preço/performance tendendo pois a enfatizar o custo de aplicação e desenvolvimento de software sobre os custos diretamente atribuídos ao hardware; assim tornou-se cada vez mais atraente a utilização de técnicas "generalizadas" que experimentaram notável evolução nos últimos 15 anos, originando uma série de classes de sistemas de software sofisticados e generalizados, uma das quais é / constituída pelos DBMS's.

2.2 - OBJETIVOS DO GERENCIAMENTO DE BANCO DE DADOS

Os principais objetivos de um DBMS são:

a) fazer com que uma coleção de dados integrados esteja disponível para uma grande variedade de usuários;
b) garantir a qualidade e integridade dos dados;
c) garantir a privacidade através de medidas de segurança / dentro do sistema;

d) permitir um controle centralizado do banco de dados, necessário para a administração eficiente dos mesmos;

e) conseguir certa "independência dos dados" no sentido de proteger o usuário de detalhes físicos de organização e armazenamento.

As seções seguintes analisam em maior detalhe cada um desses objetivos.

2.2.1 - DISPONIBILIDADE DOS DADOS

Em seu artigo [3] Everest afirma que o / maior objetivo de um DBMS é fazer com que os dados possam ser / compartilhados. Isto implica em que o banco de dados, assim como os programas, processos e modelos de simulação devam estar disponíveis a toda a comunidade de usuários daqueles dados. Há dois importantes mecanismos para tornar os dados disponíveis: a definição dos dados e o dicionário de dados [19] .

A definição dos dados é uma versão mais / sofisticada do "DATA DIVISION" do COBOL, ou do "FORMAT" do FORTRAN; em DBMS's entretanto, a definição dos dados é fornecida fora dos programas ou consultas dos usuários e precisa ser a estes adicionada de alguma maneira. A definição dos dados geralmente / consiste em um conjunto de comandos com os nomes dos elementos,/ suas propriedades (caracter ou numérico, por ex.) e suas relações com outros elementos que compoêm o banco de dados. A função de definição dos dados, quando centralizada, é de exclusiva responsabilidade do administrador do banco de dados. O programador/

ou usuário não mais possuem, portanto, o controle da maioria das relações físicas e lógicas.

Se por um lado isto restringe de certa maneira o usuário, por outro lado garante que todos os programas / usem a mesma definição; assim um novo programa pode recuperar ou atualizar os dados tão fácilmente como qualquer outro.

A definição centralizada facilita o controle de duplicação de dados, que geralmente provoca ineficiências / de armazenamento. Entretanto nem toda duplicação de dados é necessáriamente indesejável; uma duplicação controlada pode ser necessária para permitir que uma classe especial de usuários obtenha / respostas mais rápidas sem penalizar a eficiência para os demais.

O dicionário de dados é o meio pelo qual o sistema transmite à comunidade de usuários a definição dos dados. O dicionário informa o sentido do dado, seu formato, etc.., dando ao usuário uma definição precisa dos termos. Ele se encarregará / das seguintes funções: armazenagem e geração da definição dos dados, manutenção de estatísticas de uso, geração de procedimentos/ para consistência dos dados e auxiliar na segurança.

O dicionário pode também conter um coletor ou repositório de estatísticas de uso do DBMS levantadas por outros componentes do sistema. Tais estatísticas são importantes / não apenas para fins de contabilidade e apropriação de custos, como também podem ser utilizadas para melhorar a eficiência do próprio DBMS, indicando, por exemplo, reagrupamentos dos elementos / que otimizem os acessos mais frequentemente solicitados.

2.2.2 - QUALIDADE DOS DADOS

Este talvez seja o objetivo mais negligenciado de um DBMS. Dados podem ser adulterados ou destruídos devido a:

- erro humano (intencional ou acidental);
- erro de programa;
- mau funcionamento do hardware, incluindo desde perdas de alguns bits até faltas catastróficas em que se destroe todo o conteúdo de uma unidade de disco.

A manutenção da qualidade envolve a detecção do erro, determinação de como o erro ocorreu (tomando medidas que evitem a sua repetição) e a correção do dado errado. Essas operações implicam em medidas preventivas e software adicional dentro do sistema de gerenciamento de banco de dados.

2.2.3 - PRIVACIDADE E SEGURANÇA

O terceiro maior objetivo de um sistema de gerenciamento de banco de dados é a privacidade - a necessidade de se proteger o banco de dados contra acessos inadvertidos ou não autorizados. Privacidade é geralmente conseguida através de mecanismos de segurança tais como "passwords" ou chaves de privacidade. No entanto, os problemas se agravam quando o controle do sistema é descentralizado, isto é, em banco de dados distribuído, onde o fluxo dos dados supera jurisdições locais.

Uma solução para este problema é passar /

as regras de privacidade com os dados, o que é caro, mas necessário. O sistema receptor precisa então reter e fazer cumprir a privacidade original.

Auditorias de segurança são conseguidas / registrando-se os acessos (por pessoas e programas) a qualquer informação de acesso restrito. Esses mecanismos permitem que se determine quem acessou qual dado sob que condições.

2.2.4 - CONTROLE DOS DADOS

A obtenção de controle sobre os dados é um dos objetivos básicos de um DBMS. Este objetivo direciona todo o projeto do banco de dados e leva normalmente ao estabelecimento / de uma função de administração ou gerenciamento a cargo de um elemento específico: o administrador de dados. É importante notar / que o projeto do banco de dados envolve por vezes compromissos , já que usuários distintos podem ter necessidades bastante diversas e incompatíveis.

Embora a instalação de um DBMS seja um passo importante em direção a um controle efetivo de gerenciamento , o administrador de hoje enfrenta um desafio: as ferramentas disponíveis, na maioria dos DBMS's atualmente utilizados, são muito / restritivas e pouco eficientes.

2.2.5 - INDEPENDÊNCIA DOS DADOS

A estruturação dos dados pode ser descrita

a nível físico ou a nível lógico.

A estrutura física descreve a maneira como os valores dos dados estão fisicamente armazenados em um sistema. Assim são ítens associados à estrutura física: apontadores, representação de caracter, representação inteira e de ponto flutuante, complemento de um ou de dois, blocagem de registro e métodos/ de acesso.

A estrutura lógica descreve a maneira como o usuário vê os dados. Desta maneira, a DATA DIVISION de um programa em COBOL é um comando de estrutura lógica; ele lida com os nomes dos elementos e suas relações ao invés da implementação física. Um registro num programa COBOL é manipulado pelo programador sem conhecimento do hardware da máquina ou seu método de acesso.

Um sistema é dito fisicamente independente com relação aos dados, se ele pode manipular diferentes estruturas físicas ou diferentes métodos de acesso de maneira transparente ao usuário, que precisa dar informações apenas sobre a estrutura lógica. Sistemas com independência física dos dados provêem um número discreto de escolhas para a implementação do armazenamento físico dos dados, escolha esta feita a tempo de implantação em função das características do sistema e da aplicação.

Outros sistemas mais sofisticados conseguem certa independência de dados mesmo a nível lógico, permitindo que o usuário faça pedidos com pouco conhecimento da estrutura lógica dos dados. Tais sistemas podem operar corretamente mesmo / que a estrutura lógica seja, por alguma razão, alterada dentro /

de certos limites.

Os diferentes modelos de dados, atualmente em estudos, atingem uma maior ou menor independência lógica dependendo dos seus objetivos principais, no que diz respeito ao tipo/ de informação a ser armazenada, como serão recuperadas pelos usuários e principalmente quanto aos métodos de atualização e remanejamentos temporários que serão empregados pelo sistema.

Uma total independência dos dados parece , no entanto, envolver um entendimento da semantica dos dados e alguma maneira de formalizá-la.

2.3 - ELEMENTOS DE UM BANCO DE DADOS

Em sua forma mais geral, um banco de da - dos compõem-se dos seguintes elementos:

1 - Definição completa dos dados para todos os arquivos;

2 - Métodos de acesso:

   a) Indicadores (índices, elos de ligação, cadeias);

   b) Rotinas de pesquisa;

   c) Rotinas de manutenção;

3 - Os próprios arquivos de dados (arquivos públicos e in - ternos);

4 - Arquivos temporários (arquivos de trabalho, filas, / etc...);

5 - Arquivos de "backup" (dados, definição de dados, índi - ces, dados de controle, dados de segurança e programas);

6 - Tabelas de segurança (abrangendo usuários, programas, /

terminais, dados e definições de dados);

7 - Dados de controle (tais como controle de totais);

8 - Dados para auditoria;

9 - Rotinas para reinício e recuperação de dados e progra - mas;

10 - Funções especiais tais como compressão e descompressão, codificação e decodificação;

11 - Indicadores de espaço de armazenagem, mapeamento de arquivos.

Muitos dos DBMS's atuais começaram com um subconjunto desses elementos: tipicamente com uma disponibilidade limitada de definição de dados, um limitado número de métodos de acesso, e com os arquivos públicos de dados. A princípio estes / são os elementos que mais interessam ao usuário. À medida que / adquirem experiência, os fornecedores vão incluindo gradualmente/ outros elementos e os usuários vão exigindo-os.

No item 1 acima incluem-se as definições / das estruturas lógica e física dos arquivos, conforme exposto na seção precedente.

A estrutura lógica de um arquivo é a es - trutura que deverá ser levada em consideração pelo usuário, de modo a processar o arquivo com as facilidades à sua disposição e na linguagem que estiver usando. A estrutura física específica des - creve a maneira pela qual os dados são representados internamente; os usuários poderão ou não conhecer a estrutura física. O método/ de acesso é, essencialmente, a interface entre a estrutura lógica e física: é o conjunto de rotinas que permite extrair dos elemen-

tos (físicos) de armazenagem, os dados (lógicos) na forma reconhecida pelo usuário.

3 - O SISTEMA BIBLOS

3.1 - O PROJETO INICIAL

No projeto inicial, a intenção era automatizar a catalogação e o atendimento de consultas na Biblioteca / Central da UNICAMP, ou seja, permitir ao usuário consultar interativamente um banco de dados armazenado no computador da Universidade. Os documentos armazenados seriam publicações, relatórios, / livros, etc...

Bibliotecas, por sua própria natureza, / apresentam problemas complexos em recuperação de informação. Catalogação, mesmo de livros, é uma tarefa difícil de se reduzir a operações totalmente mecanizadas; no caso de publicações periódicas, o problema é ainda maior. Mas catalogação é em essência a última parte de um processo que envolve classificação, armazenamento, manutenção, renovação de assinaturas, registros de uso, etc., BIBLOS tornaria possível a criação e manutenção de um banco de dados integrando cada uma dessas funções e permitindo que todo o / processo fosse gerenciado mais coerente e economicamente.

Além do agrupamento e recuperação do catálogo da Biblioteca Central, havia também interesse em um sistema/ que possibilitasse a utilização prática de uma grande quantidade/ de informações constituída por referencias bibliográficas catalogadas periódicamente em fitas magnéticas, por diversas institui -

ções (COMPENDEX, INSPEC, etc...).

Como cada uma dessas fitas apresentasse / formatos de registros e campos particulares e não se desejasse desenvolver um sistema de recuperação particular para cada uma, decidiu-se utilizar o sistema BIBLOS para esta tarefa. Isto foi conseguido através de um conjunto de módulos de tratamento para cada serviço, os quais convertem as informações do formato original para o formato BIBLOS, integrando-as ao banco de dados.

Assim, a medida que prosseguia o desenvolvimento, ficou aparente que diversos outros projetos solicitados/ à equipe do Centro de Computação poderiam ser perfeitamente atendidos pelo sistema, tais como:

- Recuperação seletiva e emissão de etiquetas de endereços de listas de assinantes de um dado serviço;
- Recuperação seletiva conversacional de oferta ou procura de determinados serviços em congressos ou "balcões de serviço";
- Recuperação seletiva da descrição e situação de processos burocráticos existentes na seção de protocolo da universidade, envolvendo também a emissão periódica de listagens de controle.

Enfim verificou-se que a estrutura de dados / adotada para a manipulação de referencias bibliográficas é adequada a um grande número de outras aplicações.

3.2 - RAZÕES PARA A IMPLEMENTAÇÃO DE UM SISTEMA PRÓPRIO

Conforme exposto em 1.2, as diversas aplicações forçaram o sistema a se desenvolver e adaptar-se às novas/

necessidades. Esta diversidade, por outro lado, poderia sugerir / também a adoção de algum DBMS já implantado e largamente difundido no mercado. Mas as necessidades de usuários de banco de dados/ variam em uma ampla faixa, desde os que necessitam de pequena / quantidade de dados, organizados de uma maneira simples, até aqueles que requerem uma grande quantidade de informações, estruturadas de forma complexa e sofisticada.

Os sistemas de banco de dados atualmente / disponíveis tendem mais para o atendimento de usuários pertencentes ao segundo grupo. Aqueles com objetivos mais modestos em termos de sofisticação, apesar de utilizarem muitos dos recursos de um grande sistema, o fazem em uma escala menor e mais imediata. Embora necessitando de vários recursos de um DBMS, estes usuários gostariam de evitar envolvimentos com linguagens de descrição dos dados, definição de arquivos, utilização e alocação de espaço , etc...

Para estes usuários a definição do banco de dados e os recursos de manipulação do sistema são muito mais importantes que a estrutura e a eficiência das técnicas com as quais o sistema organiza e gerencia as informações. Como resultado, eles são geralmente forçados a projetar seus próprios sistemas a fim de obter simplicidade, flexibilidade e facilidade de aplicação além de se restringirem aos recursos disponíveis em sua instalação.

Foi para esse tipo de usuário que necessita de um sistema de propósito geral mas em uma escala menor e mais orientada para suas necessidades, que o sistema BIBLOS foi ini -

cialmente projetado.

BIBLOS é um sistema de banco de dados relacional implementado sob o sistema operacional TOPS-1Ø do DEC-10. O sistema foi escrito em FORTRAN e MACRO-1Ø (linguagem de montagem), exceto pelo compilador de linguagem de consulta, escrito em PASCAL. As vantagens do modelo relacional para sistemas de gerenciamento de banco de dados tem sido extensamente discutidas na literatura [4,5] .

A escolha desse modelo foi particularmente motivada pelas seguintes características:

a) o alto grau de independência de dados que esse modelo oferece;
b) a possibilidade de se prover facilidades de alto nível para consulta e atualização sem que o usuário precise explicitar / procedimentos para definição de dados, recuperação, atualização, controle de acesso e verificação de integridade.

3.3 - RECURSOS FUNCIONAIS

BIBLOS foi originalmente concebido como / um sistema que possibilitaria buscas lógicas e recursos de seleção de dados mantidos em um sistema iterativo e "on line". Era esperado que muitos dos usuários não seriam programadores. Portanto BIBLOS foi projetado para oferecer os seguintes recursos funcionais:

- Estabelecer o formato dos dados e as estruturas do banco de dados;
- Adicionar e retirar dados e modificar dados já inseridos;

- Recuperar dados de uma maneira seletiva e mostrar os resulta - dos;
- Gerar relatórios utilizando formatos especificados pelo usuário;
- Prover facilidades para análise de dados e operações sobre os mesmos.

A maneira pela qual esses recursos seriam/ apresentados ao usuário foi ditada pelos seguintes objetivos:

- Sistema completamente iterativo com todas as facilidades disponíveis via terminais;
- Simples e fácil de se usar, incluindo uma sintaxe de comandos / utilizando mnemonicos;
- Não orientado para ou limitado a tipos específicos de aplica - ções;
- Investimento mínimo em termos de instalação de sistema, treinamento e esforço de implementação.

3.4 - ESTRUTURA DOS DADOS

Neste ponto do texto serão introduzidos , para os leitores não familiarizados, alguns conceitos gerais em - pregados na teoria do modelo relacional.

Em uma série de artigos [4, 11, 12] , Codd introduziu o modelo relacional de dados e discutiu suas vantagens em termos de simplicidade, simetria, independência dos dados e de completeza semantica ("semantic completeness").

Dados os conjuntos D_1, D_2, ..., D_n (não necessáriamente distintos), R é uma relação nesses n conjuntos, se

R é um conjunto de elementos da forma $(d_1, d_2, \ldots, d_n)$ onde / $d_j \in D_j$;para cada j= 1, 2, ..., n. Ou seja, R é um subconjunto / do produto cartesiano $D_1 x\ D_2 x \ldots x\ D_n$. D_j é o j-ésimo domínio de R. O valor n é chamado de grau de R. Os elementos de uma relação/ de grau n são chamados n-tuplas ou simplesmente tuplas.

A figura 3.1 ilustra uma relação chamada / ITENS definida sobre os domínios ITEM (identificação do item), NOME (nome do item) e COR (cor do item)

ITENS

NOME	*ITEM*	*COR*
caneta	*I1*	*azul*
lapis	*I2*	*preto*
regua	*I3*	*branca*
borracha	*I4*	*verde*
lapis	*I2*	*azul*

Fig. 3.1 - Relação ITENS

Como mostra a figura 3.1, é conveniente representar uma relação como uma tabela, onde cada linha representa uma tupla.

Utilizar-se-á para representar uma relação, uma tabela com as seguintes características:

1 - Cada linha representa uma tupla de T;

2 - Não importa a ordem das linhas;

3 - Todas as linhas são distintas;

4 - A ordem das colunas é significante já que corresponde à ordem $D_1, D_2, \ldots, D_n$ dos domínios sobre os quais R está definida.

No entanto, é possível designar qualquer / coluna em particular pelo nome do domínio correspondente, em lu - gar de pela sua posição relativa.

Desta forma pode-se afirmar que:

5 - Não importa a ordem das colunas, desde que estas sejam convenientemente rotuladas.

Entretanto, nem sempre o nome do domínio é um rótulo adequado para as colunas: como ilustrado na figura 3.2, duas colunas podem pertencer a um mesmo conjunto.

COMPONENTE

PEÇAS	PEÇAS	QUANTIDADE
1	5	9
2	5	7
3	5	2
2	6	12
3	6	3
4	7	1

Fig. 3.2 - Relação "COMPONENTE" definida sobre os domínios (PEÇAS, PEÇAS, QUANTIDADE)

Na relação COMPONENTE da fig. 3.2, cada / tupla (x, y, z) indica que z unidades da peça x são necessárias para se compor uma peça y. Ou seja, nem todos os domínios de R são distintos. Nestes casos, para manter a distinção entre as duas colunas que tem valores pertencentes ao conjunto PEÇAS, usa-se um nome distinto para cada uma delas: esse nome é o nome do atributo. No exemplo acima, poder-se-ia atribuir à primeira coluna o nome de atributo PEÇA1 e à segunda PEÇA2. Dessa forma a relação COMPONEN- TE tem os seguintes nomes de atributos (PEÇA1, PEÇA2, QUANTIDADE)

enquanto que os domínios correspondentes são (PEÇAS, PEÇAS, QUANTIDADE).

Nomes de atributos são uma maneira de proteger o usuário da necessidade de conhecer as posições relativas/ dos domínios. Cada uso distinto de um dos domínios do banco de dados em uma relação R é chamado de atributo de R.

Até agora, foram mostrados exemplos de relações definidas em domínios simples - domínios cujos elementos / são valores atomicos.

Mas pode-se definir também relações onde os elementos dos domínios são em si relações, e essas relações podem também ser definidas sobre outras e assim por diante. Por / exemplo, um dos domínios de uma relação que contenha diversas informações sobre empregados de uma firma, pode ser o "histórico de salários". Um elemento desse domínio é a relação binária definida sobre os domínios DATA e SALARIO.

Uma relação é dita estar na forma normalizada se cada um dos seus domínios é simples, isto é, nenhum domínio é em si uma relação [21].

3.4.1 - ARQUIVOS DE DADOS

A representação lógica dos arquivos de dados em BIBLOS está baseada no modelo relacional. Cada arquivo do banco de dados representa uma única relação do modelo, como um / conjunto homogêneo de n-tuplas.

Na terminologia utilizada, cada n-tupla é

denominada unidade ou registro e os atributos são os campos de informação em que se divide uma unidade. No texto, os dois tipos de terminologia serão usados indistintamente.

Na implementação física utilizada, cada relação corresponde a um arquivo em disco constituído por cadeias / concatenadas de caracteres, cada cadeia representando uma tupla.

A cada tabela ou relação está associado um arquivo especial denominado "controle" que indica a localização/ (no arquivo que representa a relação) da tupla, dada a sua identificação interna (que é um inteiro atribuído à tupla quando de sua introdução na tabela). O i-ésimo registro do controle tem a seguinte estrutura lógica:

pos	tam

onde:

POS - apontador para o local onde está armazenada a tupla com / identificação interna i.

TAM - número de caracteres que compõem a tupla.

A figura 3.3 ilustra a relação entre um arquivo de dados e seu controle.

3.4.2 - ARQUIVOS DE INVERSÕES

Embora buscas sequenciais em tabelas sejam permitidas, em muitos casos é desejável dispor de um método de / acesso mais eficiente. Para tanto podem ser criados arquivos invertidos ou índices. Um arquivo invertido de um determinado campo

de informação A é formado pela sequencia de pares do tipo (pal , apt) onde pal é um valor do campo A e apt é um apontador para uma lista de identificações internas das tuplas cujo campo A tem o valor pal.

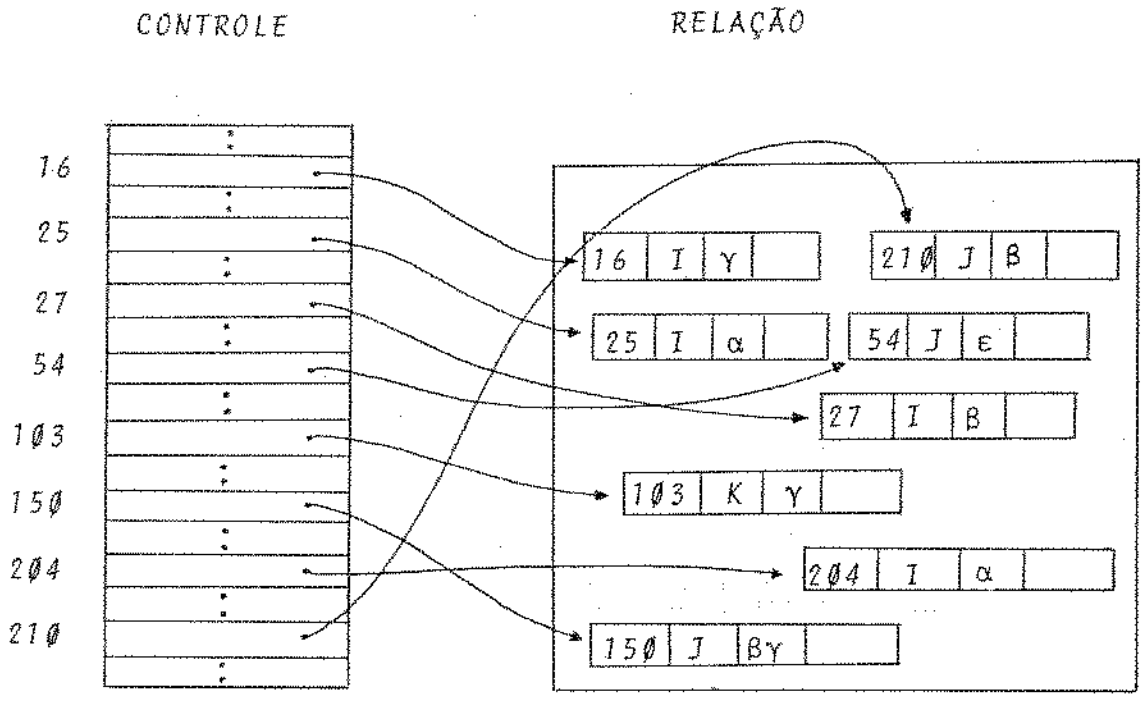


Fig. 3.3 - Relação e seu arquivo de controle

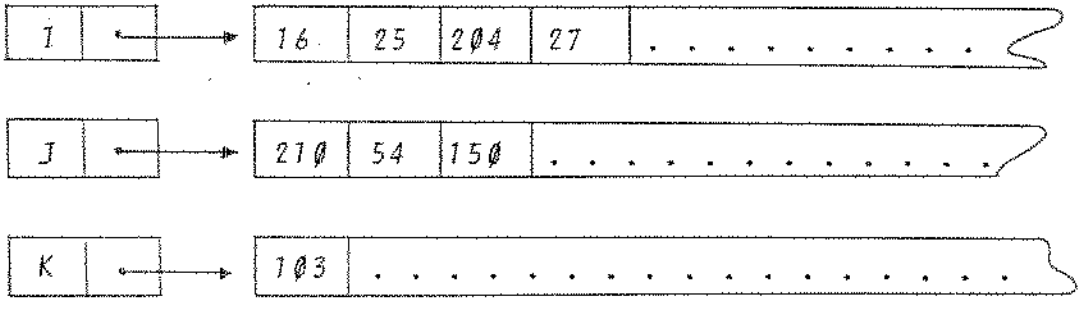


Fig. 3.4 - Arquivo de inversões para o primeiro campo da relação da figura 3.3.

O arquivo invertido será construído em ordem alfabética para o campo pal [fig. 3.4] , de forma a permitir/ buscas binárias.

3.4.3 - CATÁLOGOS DO SISTEMA

Em DBMS, catálogos são arquivos contendo / parâmetros que caracterizam uma determinada aplicação do sistema, mostrando a situação atual dos seus arquivos de dados, índices e controles.

Foram implementados os seguintes catálogos para os módulos de manutenção, recuperação e controle a serem descritos em 3.6.

a) Catálogo de relações: contêm uma entrada para cada relação do banco de dados; cada entrada contêm as seguintes informações:
   - rel-nome: identificação externa da relação-identificador a ser utilizado pelo usuário para designar a relação;
   - rel-id: identificação interna da relação a qual corresponde/ a entrada. É uma cadeia de 3 letras que permite construir o nome do arquivo em que está armazenada a relação;

   As duas informações acima constituem campos-chave neste catálogo.
   - criador: dono da relação;
   - proteção: indica a proteção aplicável a esta relação, isto é, quais as operações permitidas sobre esta relação para os diversos tipos de usuários;
   - ureg: número atual de tuplas armazenadas na relação;

- natr: número de atributos contidos na relação;
- max: tamanho em caracteres da maior tupla armazenada na relação;
- ublkc: último bloco ocupado do arquivo de controle da relação;
- ucarac: posição do último caracter da última tupla armazenada na relação;
- ublkm: número do último bloco de disco ocupado pela relação;
- ninv: número de atributos invertidos;
- AT1, AT2, ..., ATN: identificações internas dos atributos invertidos.

b) Catálogo de atributos: contém uma entrada para cada atributo / de cada relação do sistema; cada entrada contém as seguintes / informações:

- atr-nome: identificação externa do atributo-identificador a ser utilizado pelo usuário para designar o atributo.
- atr-id: identificação interna do atributo

  Formato: SSS99 onde

  SSS - identificação interna da relação a qual pertence a atributo.

  99 - identificação interna do atributo dentro da relação a qual pertence. (inteiro com 2 digitos).

- tipo: código que identifica o tipo e formato do atributo.

  Formato: $d_{\emptyset}$ d_1 d_2 onde

  $d_{\emptyset} = \emptyset$ - ASCII

  1 - SIXBIT

2 - binário

d_1 = Ø - fixo

1 - variável

d_2 = Ø - numérico

1 - alfanumérico

2 - alfabético

O campo d_2 não se aplica caso $d_Ø$ = 2.

- comp: comprimento do campo se fixo, ou limite máximo se variável.

OBS: Note-se que atr-nome não é chave neste catálogo: um atributo de mesmo nome pode aparecer mais de uma vez desde que em relações diferentes, possivelmente com tipo diferente/ em cada relação. Neste caso, porém, haverá necessidade de um procedimento de conversão entre os dois tipos para permitir comparações de valores do atributo nas diversas relações em que aparece.

3.5 - MÉTODOS DE ARMAZENAMENTO E ACESSO

A organização física da informação dentro/ de um banco de dados tem grande influência no custo e desempenho/ de um sistema de informação computarizado. Uma organização eficiente para um problema específico depende do volume e da estrutura intrínseca da informação armazenada, da frequência e extensão/ das atualizações, dos formatos e frequência de recuperações e das características de acesso do hardware [6] .

O processo do projeto de um banco de dados

procura determinar uma organização de dados de forma a otimizar / algum critério de desempenho sujeito a algumas restrições.

Quatro fatores básicos definem o problema/ do projeto:

1) Os dados a serem armazenados - seu tamanho, volume e volatilidade;
2) O equipamento de armazenamento - seu custo de operação e características de acesso;
3) A recuperação dos dados - sua frequência, prioridades, tempos/ de resposta exigidos, critérios de seleção;
4) O objetivo do sistema - uma expectativa de desempenho condicionada a um conjunto de restrições do projeto operacional.

3.5.1 - EQUIPAMENTO DE ARMAZENAGEM

Os objetivos na organização de um banco de dados diferem substancialmente dependendo do lugar onde estão os dados a serem acessados e atualizados. Se todos os dados estão disponíveis já de início na memória principal, então o objetivo é simplesmente minimizar o número médio de instruções de máquina a serem executadas durante as operações do processamento dos dados.

No entanto, quando os dados estão armazenados na memória secundária, a velocidade de acesso é menor em várias ordens de magnitude, e a principal preocupação é reduzir os tempos de acesso.

O programador inexperiente irá subestimar/ o tempo necessário para se completar uma operação de entrada/saí-

da (10-1000 ms) em relação ao tempo necessário para se executar / uma instrução do computador (0.1 - 10 μs).

Assim uma das maiores preocupações no projeto de um sistema de gerenciamento de banco de dados é reduzir o número de operações de entrada/saída necessário e alguns dos métodos mais frequentemente usados incluem a redução dos custos fixos de acesso aos dados, aumento do volume de dados transferidos em cada acesso ("bufferização") ou aumento do conteúdo da informação transferida (codificação, compactação, compressão, etc...).

3.5.2 - TÉCNICAS DE RECUPERAÇÃO DE REGISTROS

Antes de se passar diretamente à descrição dos métodos de acesso e armazenamento empregados no projeto BI - BLOS é conveniente, nesse ponto, analisar os métodos utilizados e recomendados na literatura de acordo com alguns tipos característicos de aplicações.

Essa análise baseia-se num estudo feito / por Severance e Carlis [6].

O número de registros que é normalmente requisitado no processo de recuperação de um banco de dados caracteriza uma dada aplicação.

Assim diz-se que uma aplicação pode se caracterizar por recuperações de grande número de registros (ex: folha de pagamento), recuperação de pequenos conjuntos de registros e recuperação de registros individuais.

Essa divisão determina para cada tipo uma/

série de métodos de armazenamento e acesso aconselháveis.

No projeto do BIBLOS a preocupação inicial era armazenar o maior número possível de dados ocupando o mínimo/ possível de espaço na memória secundária. Optou-se então por uma estrutura compactada, onde os registros estão divididos em campos de tamanho variável. Para dinamizar a recuperação, implementou-se o método de listas invertidas. Embora a experiência confirme que a estrutura escolhida possibilita recuperações rápidas, deseja-se garantir que esse comportamento se manterá em aplicações as mais diversas.

Para tanto, passar-se-á ao exame das vantagens e desvantagens dos vários métodos em função do tipo de recuperação.

3.5.2.1 - RECUPERAÇÃO DE GRANDES SUBCONJUNTOS DE REGISTROS

Aplicações tradicionais de processamento / de dados tais como folha de pagamento e faturamento são tarefas / que tipicamente processam uma grande quantidade de registros em / um banco de dados. Há 3 razões principais porque uma estrutura de armazenamento sequencial suporta eficientemente tais aplicações:

1) A organização dos dados é simples: o "overhead" de software e estrutura são mínimos, o "backup" e os procedimentos de recuperação são imediatos. Um banco de dados organizado sequencial - mente é tipicamente mais barato que a implementação de uma organização baseada em listas;
2) A organização dos dados aproveita as características de acesso das memórias secundárias. Já que é conhecida a sequência dos

registros a serem recuperados, grupos de registros sucessivos/ podem ser blocados para serem recuperados num único acesso;

3) O banco de dados pode ser armazenado em fita magnética, o que/ oferece uma vantagem de custo de armazenamento de 50:1 em relação a equipamentos de acesso direto.

A despeito de suas vantagens, esse tipo de organização possui duas limitações significativas que impedem seu uso em muitos sistemas. Primeiro é o alto custo de atualizações / individuais. Por exemplo, se um registro precisa ser adicionado a um bloco de registros, então o efeito de um "overflow" nesse bloco pode se estender a outros blocos e assim por diante. Isso pode implicar na necessidade de se reescrever todo o arquivo. Esse problema pode ser parcialmente resolvido deixando-se em cada bloco / um espaço livre para a adição de novos registros. Ou pode-se adicionar uma área de "overflow" para cada bloco a medida que o espaço for sendo exigido.

Mesmo assim o processamento pode se tornar proibitivo se as atualizações forem frequentes ou se conjuntos de registros são inseridos.

A manutenção de arquivos sequenciais só é prática quando as atualizações podem ser acumuladas durante um período de tempo e depois intercaladas no banco de dados em reorganizações periódicas ou mantidas em um arquivo pequeno diferencial [28] separado e logicamente incorporado ao arquivo principal / quando este for processado.

Portanto uma estrutura de arquivo sequencial é imprópria para bancos de dados voláteis que necessitam /

atualizações imediatas. Esse tipo de estrutura é também limitado/ quando se deseja recuperar pequenos conjuntos de registros. En / quanto, algumas vezes, é possível agrupar vários registros em sub conjuntos em um banco de dados sequencial, todo o arquivo precisa geralmente ser pesquisado para se recuperar qualquer subconjunto/ de registros.

Dessa maneira, se o subconjunto é pequeno, existem estruturas alternativas que permitem uma recuperação mais eficiente, conforme exposto a seguir.

3.5.2.2 - RECUPERAÇÃO DE PEQUENOS SUBCONJUNTOS DE REGISTROS

Estamos assumindo como limite entre um pequeno e grande subconjunto de registros, aproximadamente 10% do banco de dados. A porcentagem precisa para um problema específico depende das velocidades relativas entre o acesso direto e o acesso sequencial a registros. Suponha-se que todo um banco de dados/ de n registros pode ser lido sequencialmente em, por exemplo, s / segundos, enquanto a recuperação individual gasta em média t se - gundos por registro. Não há vantagem alguma na "recuperação direta" de um conjunto de registros cujo tamanho exceda 100 s/nt % do banco de dados. E ainda, o "overhead" e complexidade do software inerente para suportar esse acesso direto é justificado apenas / quando o banco de dados em si é relativamente grande e a velocidade de recuperação é essencial.

Em um sistema de recuperação de documentos são típicas as aplicações nas quais se deseja recuperar rápida -

mente uma pequena quantidade de registros de um grande banco de dados. A figura 3.5 ilustra quatro tipos de estruturas de arquivos, aconselhãveis para suportar tais aplicações.

Em cada um deles, um conjunto de registros $Q = \{Q_1, Q_2, Q_3, Q_4, \ldots\}$ deve ser recuperado e cada estrutura utiliza apontadores para definir meios de acesso que pulam áreas irrelevantes do arquivo. Duas propriedades essenciais distinguem as alternativas:

1 - A informação sobre a estrutura entre os registros está armazenada dentro da área dos registros, como uma lista, ou está fisicamente separada desta área como uma lista invertida;

2 - A estrutura de apontadores endereça registros ou células individuais (células são coleções de registros transferidas da memória principal como uma única unidade).

O método de listas invertidas de registros será examinado mais detalhadamente já que é o método empregado no sistema BIBLOS.

Listas invertidas são mais populares na prática, por possibilitar recuperação direta de conjuntos de registros de um banco de dados. Pelo fato de a informação da estrutura estar fisicamente separada da área do registro, esta organização/ oferece várias vantagens operacionais:

1 - Pode-se escolher melhor a estrutura do registro sem se preocupar com os meios de acesso empregados. Ainda mais, o projeto/ de estrutura de registro não é afetada pelas subsequentes inclusões (ou remoções) de listas invertidas;

2 - Desde que os apontadores não são armazenados nos registros, o

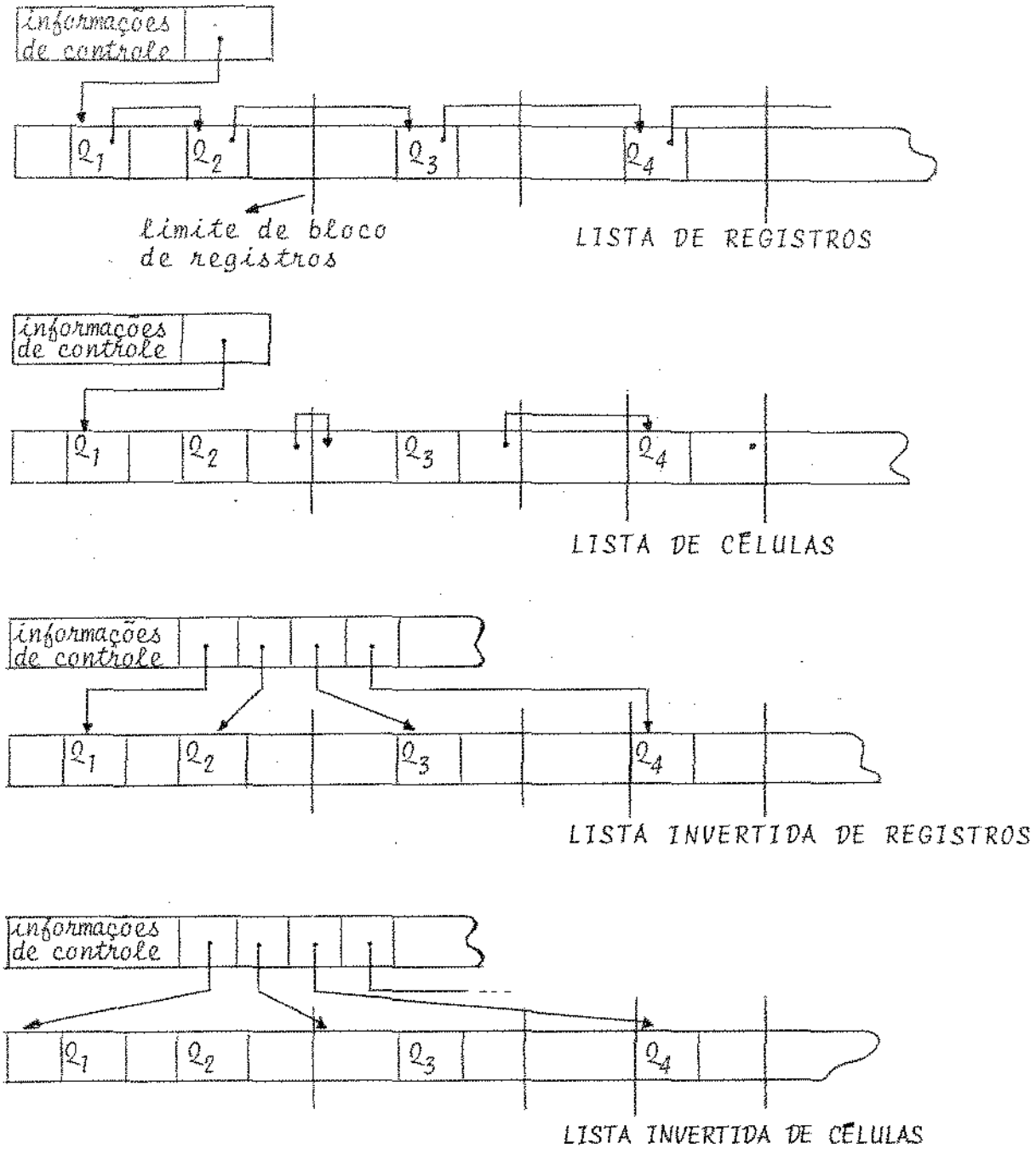


Fig. 3.5 - Métodos de acesso baseados em apontadores

número destes, em cada célula, é maior. Assim sendo, o número de células acessado durante uma recuperação é em geral menor;

3 - Comparando com a organização de multi-listas [7], o "overhead" total de espaço usado em apontadores é geralmente reduzido : quando apontadores de várias listas são armazenados nos registros é necessário utilizar um de dois esquemas possíveis: a) um campo fixo de apontador aparece em cada registro para cada lista; b) utiliza-se um campo variável onde cada apontador é identificado por um rótulo. O espaço necessário em qualquer / destas opções é maior que o utilizado no sistema de listas invertidas;

4 - Listas invertidas podem aumentar a velocidade das operações / de recuperação. Elas podem ser transportadas rapidamente para a memória principal a fim de servir a duas funções:

- sistemas de gerenciamento de dados podem usar listas invertidas a fim de executar uma "bufferização" antecipada, e quando a ordem de recuperação é imaterial ao usuário do sistema, uma análise da lista invertida pode organizar a recuperação para reduzir o tempo total de acesso;
- no processamento de uma recuperação atendendo a vários critérios, operações booleanas entre conjuntos podem ser executadas rapidamente com os apontadores de várias listas invertidas para se definir dinamicamente um novo subconjunto de registros.

Quando comparadas à organização sequencial, todas as organizações acima possuem várias desvantagens: o soft - ware necessário para a recuperação é mais complexo; a recuperação geralmente necessita de mais de um acesso à memória secundária pa

ra cada registro recuperado; o espaço ocupado pelos apontadores / constitue um "overhead" de armazenamento; e finalmente, o gerenciamento e a manutenção dos dados para um grande conjunto de listas representam um problema de projeto de banco de dados cuja solução afetará criticamente o desempenho total do sistema. No entanto, quando é essencial uma recuperação rápida de pequenos subconjuntos de registros de um grande banco de dados, uma dessas estruturas de arquivos precisa ser empregada. A estrutura de lista/ invertida é geralmente preferida.

3.5.2.3 - RECUPERAÇÃO DE REGISTROS INDIVIDUAIS

Para a recuperação de registros de um banco de dados pode ser utilizada uma série de estruturas e técnicas diferentes. Dois extremos são representados pela consulta sequencial ("scanning") e pelo endereçamento direto. Se o projetista do sistema dispõe de memória suficiente para alocar uma posição de armazenamento para cada valor possível do identificador do registro (chave), então é sempre possível empregar uma função de endereçamento direto que converterá cada valor do identificador em um único endereço de armazenamento. Qualquer registro armazenado pode assim ser recuperado com um único acesso. Embora a técnica seja rápida, suas limitações são óbvias: se, por exemplo, os registros de empregados fossem identificados pelo número de CPF, então apenas alguns dos milhões de registros alocados conteriam dados / realmente.

O tempo da busca sequencial pode ser redu-

zido consideravelmente, se os registros estiverem fisicamente ordenados pelos valores de seu identificador, empregando-se a técnica de busca binária [6] .

Por outro lado, sequências de comparações/ entre registros podem ser definidas explicitamente em arquivos fisicamente desordenados. Para tanto armazena-se em cada registro de dados, apontadores que indicam os registros sucessores em estruturas de árvores. Vários algoritmos de buscas em árvores foram analisados por Knuth[7] .

No entanto, para grandes bancos de dados / existem outras técnicas de busca mais simples e eficientes.

O tempo de busca de registros em disco é afetado em pequena escala pelo tempo de execução de instruções de máquina; o fator dominante que determina a duração de uma busca é a velocidade com que os registros são trazidos para a memória / principal. Assim, quando os registros são grandes é mais rápido / acessar registros menores de um arquivo de índices. Por este motivo, diversos esquemas de indexação foram sugeridos e tem sido utilizados: uma maneira é colocar-se em cada registro do arquivo de índices ("full index") o identificador do registro e a sua localização no arquivo de dados correspondente.

Mas o tempo necessário para se pesquisar / qualquer arquivo seja de dados ou de índices, é consideravelmente reduzido se os seus registros estiverem ordenados pelo valor do identificador e blocados: isso porque pode-se criar um outro tipo de índice ("block index") onde cada registro contém um apontador/ para o bloco do arquivo de dados, e o intervalo de chaves dos re-

gistros que podem ser encontrados nesse bloco. Se esse arquivo de índices se tornar muito grande, essa técnica pode ser reaplicada/ construindo-se índices em vários níveis ("hierarchic block index").

Muito embora a velocidade de recuperação / através de índices seja suficiente em muitas aplicações de banco/ de dados, há importantes sistemas de informação (como reserva de passagens) cuja grande densidade de consulta requer acesso ainda/ mais rápido. Para as aplicações desse tipo, onde o endereçamento/ direto é impraticável, tem sido desenvolvidas técnicas de endereçamento conhecidas como algoritmos de "hashing", onde o identificador de registro (chave) é transformado no endereço de memória / usado para o armazenamento e recuperação desse registro [6].

As técnicas de endereçamento direto e "hashing", ditam as posições físicas onde serão armazenados os registros, portanto não podem ser aplicadas em sistemas onde a localização dos registros é decidida por outros critérios. Nestes casos as técnicas acima somente podem ser utilizadas se modificadas pela introdução de uma tabela ("scatter-table") construída da seguinte maneira: à medida que um novo registro é armazenado no banco de dados (de alguma maneira convencional) um apontador para / ele é armazenado na tabela em um endereço calculado através da transformação do identificador do registro (seja por endereçamento direto-neste caso denomina-se a esta técnica: "scatter address" seja por "hashing"-caso em que recebe o nome de "scatter hashing"). Esse apontador será usado sempre que se quiser recuperar o registro pelo seu identificador. Assim sendo, essa estrutura ("scatter address") corresponde ao arquivo de controle (ver 3.4.1) utiliza-

do em BIBLOS.

3.5.2.4 - AS TÉCNICAS DE RECUPERAÇÃO UTILIZADAS NO BIBLOS

Conforme já exposto em 3.4.1, no BIBLOS os registros são armazenados sequencialmente nas tabelas a que pertencem, mas a fim de facilitar o acesso individual a cada um deles é criado juntamente com a tabela um arquivo de controle ("scatter address") onde cada entrada contém um apontador para um determinado registro da tabela correspondente. Esse arquivo de controle pode ser acessado diretamente, já que suas entradas são de tamanho fixo.

Assim sendo, os registros podem ser acessados em uma de três maneiras: a) sequencialmente, na ordem em que estão armazenados; b) na ordem de entrada no arquivo "controle", isto é, em ordem de identificação interna e c) individualmente através de uma busca no controle, dada a sua identificação interna.

Uma quarta maneira de se acessar os registros é através das listas invertidas (vide 3.4.2) que fazem referencia às identificações internas dos registros. As listas invertidas, na verdade, agrupam alguns registros de acordo com algum valor que, em determinado campo, eles possuam em comum. Esta é uma maneira de se agrupar registros ("clusters") segundo alguma característica comum, no caso, os termos que compõem os campos de informação.

As recuperações pelo sistema BIBLOS serão

permitidas através de comandos na linguagem SEQUEL ou através de programas de usuário.

No primeiro caso, os registros a serem recuperados serão determinados pelo comando de entrada e cabe ao interpretador do módulo de recuperação (vide seção 3.6.2) decidir / para cada situação, qual o método (sequencial, listas invertidas, ou busca no controle) a ser utilizado.

No caso de programa de usuário, BIBLOS coloca à disposição, operadores pertencentes ao módulo de acesso / (vide seção 3.6) que permitem a recuperação de registros dos diversos arquivos do sistema cabendo ao usuário escolher as estruturas e métodos mais convenientes para sua aplicação.

Em trabalho recente [9] , o desempenho da recuperação em vários sistemas foi estudado considerando-se o número de registros recuperados em função do número de atributos / desses registros. Os resultados estão resumidos na tabela apresentada na fig. 3.6 que dá, para cada caso, a técnica de armazenagem aconselhada, e entre parênteses a respectiva técnica de busca. Por exemplo, quando uma aplicação requer a recuperação de todos / os atributos para todos os registros, é aconselhável percorrer um arquivo sequencial. Quando apenas poucos registros são necessários mas aproximadamente todos os seus atributos, então é melhor/ usar uma estrutura de arquivo sequencial com listas invertidas. Para aplicações que requerem apenas poucos atributos independente do número de registros, arquivos transpostos são aconselhâveis [9] . Um arquivo transposto de uma relação é um arquivo em que estão armazenados sequencialmente todos os valores do pri-

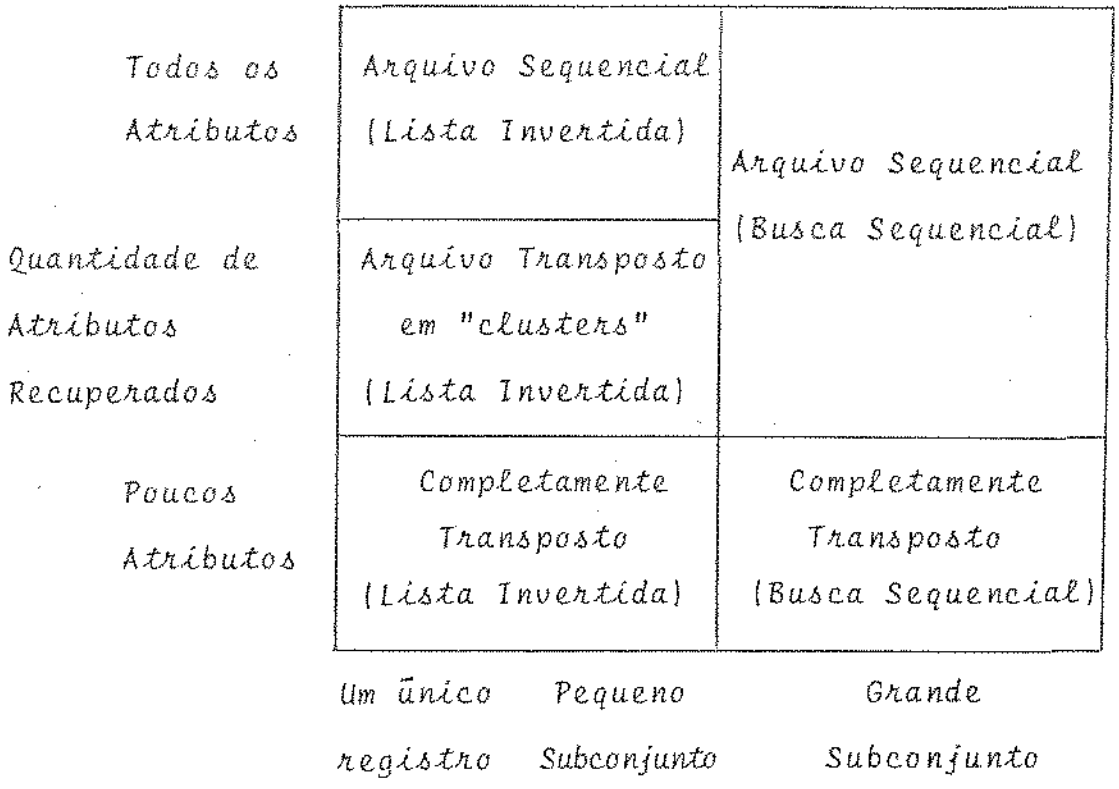


Fig. 3.6 - Compromisso entre a quantidade de registros recuperados e quantidade de atributos

meiro atributo de cada tupla, seguidos, na mesma ordem de tuplas, dos valores de segundo atributo, e assim por diante.

Quando BIBLOS foi inicialmente projetado / visava servir principalmente, como já foi dito, o serviço de consulta em uma Biblioteca. A estrutura de armazenagem sequencial com listas invertidas implica em obter melhor desempenho na recuperação de poucos registros com todos ou quase todos seus atributos o que satisfaz aos objetivos iniciais.

Permitindo-se, por outro lado, buscas se - quenciais nas relações é possível atender eficientemente a aplica

ções que envolvem recuperações de grande número de registros com muitos de seus atributos.

É importante ressaltar que a estrutura de arquivos transpostos [9] é aconselhada somente quando poucos / atributos devem ser recuperados em relação ao número de registros, atingindo seu pior desempenho quando o número de atributos é muito maior se comparado ao número de registros.

3.5.3 - ARMAZENAGEM DOS CAMPOS DE INFORMAÇÃO NOS REGISTROS

Os registros ou tuplas armazenadas no sistema BIBLOS são de tamanho variável, precedidos pelo seu identificador interno, atribuído pelo sistema.

Considere-se um registro com dois campos: o primeiro é identificado como NOME (isto é, identificação externa ou atr-nome, ver 3.4.2) e tem como valor "João" e o segundo é identificado por TELEFONE e tem como valor "526942". A figura 3.7 ilustra a estrutura de armazenamento utilizada.

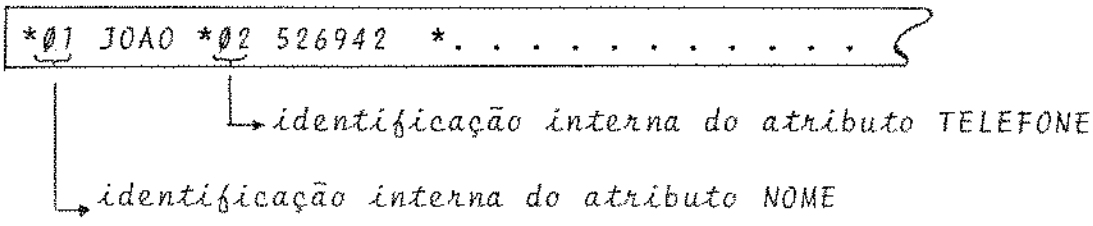


Fig. 3.7 - Estrutura de Armazenamento dos Campos de Informação

Cada campo é identificado ou rotulado por um inteiro distinto. Um caracter reservado, denotado por asterisco ("*") na figura 3.7 / permite a delimitação de cada campo.

A maneira como os dados são armazenados / nas relações permite que os campos apareçam em qualquer ordem com tamanho (ou comprimento) variável; se, para algum registro, o campo é nulo, ele pode estar totalmente ausente.

Há vantagens e desvantagens em se adotar / uma estrutura de campos rotulados ao invés de distinguir cada campo por sua posição (ordem) no registro: se por um lado o rótulo / representa um "overhead" que, no caso de campos muito pequenos pode ser significativo, por outro lado sua presença evita desperdiçar espaço com brancos ou outro marcador para denotar campos nulos. A utilização de rótulos permite ainda que os campos apareçam em qualquer ordem, possivelmente simplificando algoritmos de manutenção do sistema.

3.6 - ORGANIZAÇÃO GERAL

O sistema BIBLOS constitui-se basicamente/ de 4 partes:

- módulo de acesso
- módulo de manutenção
- módulo de recuperação
- módulo de controle

Estas quatro partes estão relacionadas entre si conforme mostra a figura 3.8.

O módulo de recuperação coloca à disposição do usuário uma linguagem de consulta de alto-nível denominada SEQUEL2 [24], pela qual ele interage com o banco de dados sem que se precise ter conhecimento da estrutura interna dos arquivos.

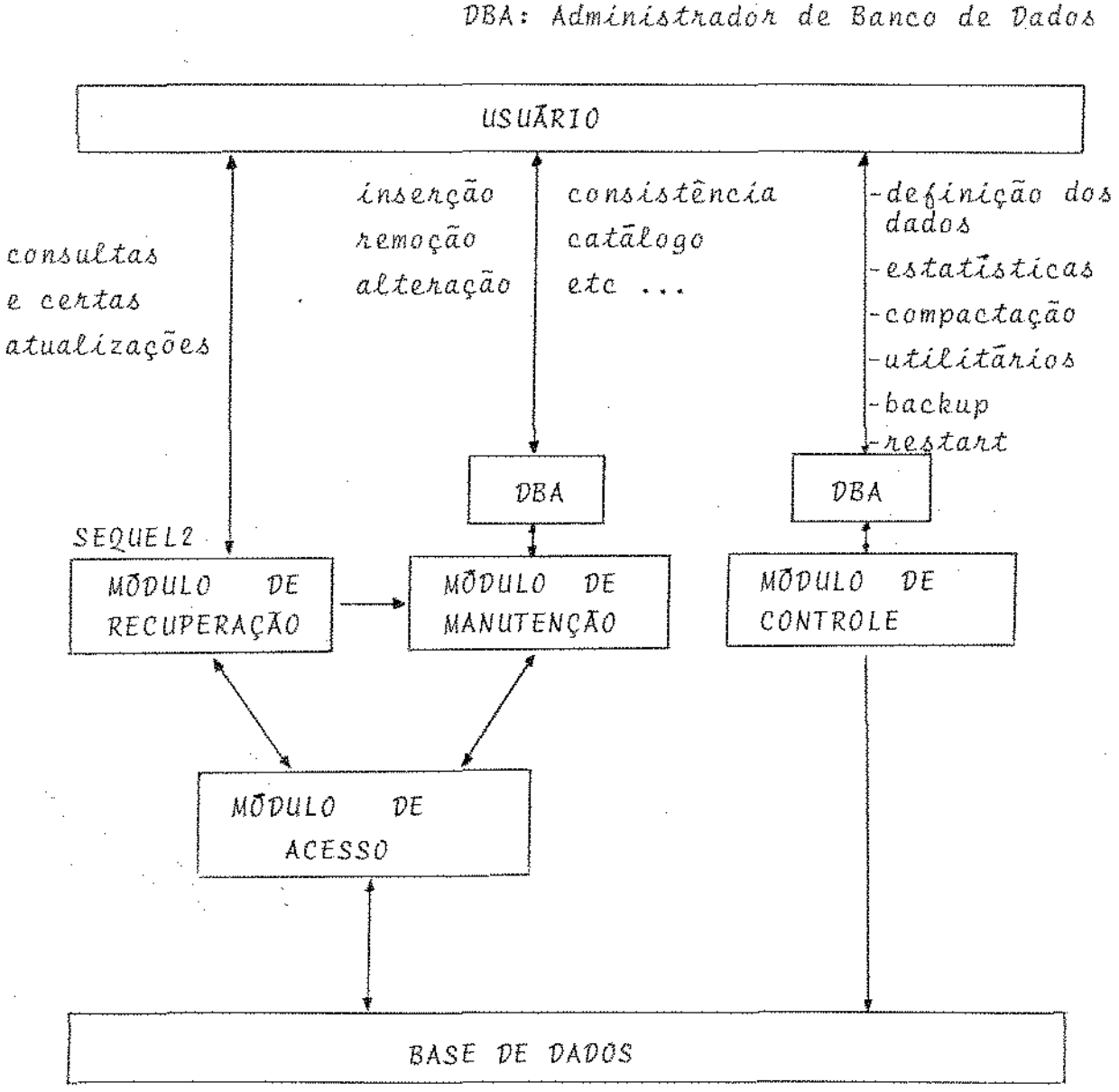


Fig. 3.8 - Os módulos constituintes do BIBLOS

O módulo de manutenção se incumbe da consistencia e controle de qualidade dos dados a serem incorporados ao DB; da inserção, remoção e alteração das informações do DB; da criação e remoção das tabelas, índices e catálogos do sistema.

O módulo de acesso é constituído por um conjunto de rotinas que cuidam do acesso aos registros e aos índices das tabelas no nível mais interno. Esse acesso é feito registro a registro. Os operadores disponíveis são os de inserção, remoção ou busca de registros individuais dada a sua identificação/ interna e a tabela a qual pertencem. A busca em arquivos invertidos é feita fornecendo o índice, o campo ou atributo invertido e a tabela correspondente, sendo então devolvida, caso haja sucesso na busca, uma lista das identificações internas dos registros correspondentes.

O módulo de controle cuida da definição / dos dados nos catálogos do sistema; da emissão periódica de relatórios de estatísticas de erros e consultas; da manutenção dos arquivos-diário ("logfiles"); da compactação e reaproveitamento dos espaços inutilizados pelo módulo de manutenção; dos utilitários / de consulta ao DB e finalmente do "backup" e reinicialização do / sistema na ocorrencia de erros de máquina ou de programas.

As seções seguintes descrevem mais detalhadamente os módulos de manutenção e recuperação.

3.6.1 - MÓDULO DE MANUTENÇÃO

Esse módulo é constituído de um conjunto /

de programas que tratam da atualização das relações, índices e catálogos, bem como da consistência dos dados de entrada.

A inserção de novas tuplas em uma determinada relação é feita pelo programa BLBINS: as novas tuplas são / fornecidas num formato padrão em um arquivo denominado INSERE / acessado tanto pelo usuário como pelo módulo de recuperação.

Formato: ⌊tid⌋ ⌊rel-id⌋ ⌊atr-id⌋ ⌊c⌋ ⌊conteudo⌋

onde

- tid: identificação interna da tupla a ser inserida. Quando tid é igual a Ø significa que essa inserção está sendo feita / pelo usuário e a essa nova tupla o sistema deve atribuir / uma nova identificação interna.
  Caso tid seja diferente de Ø significa que essa inclusão é na verdade a introdução de uma nova versão dessa tupla feita pelo módulo de recuperação (vide fig. 3.9).
- rel-id: identificação interna da relação (máximo de três caracteres).
- atr-id: identificação interna do atributo dessa tupla a ser inserido.
- c: código de continuação do conteúdo do campo.
- conteúdo: conteúdo do campo a ser inserido.

A remoção de tuplas em uma determinada relação é feita pelo módulo BLBDEL; as identificações internas das tuplas a serem removidas são fornecidas num formato padrão pelo arquivo DELETE criado pelo módulo de recuperação (vide fig. / 3.1Ø).

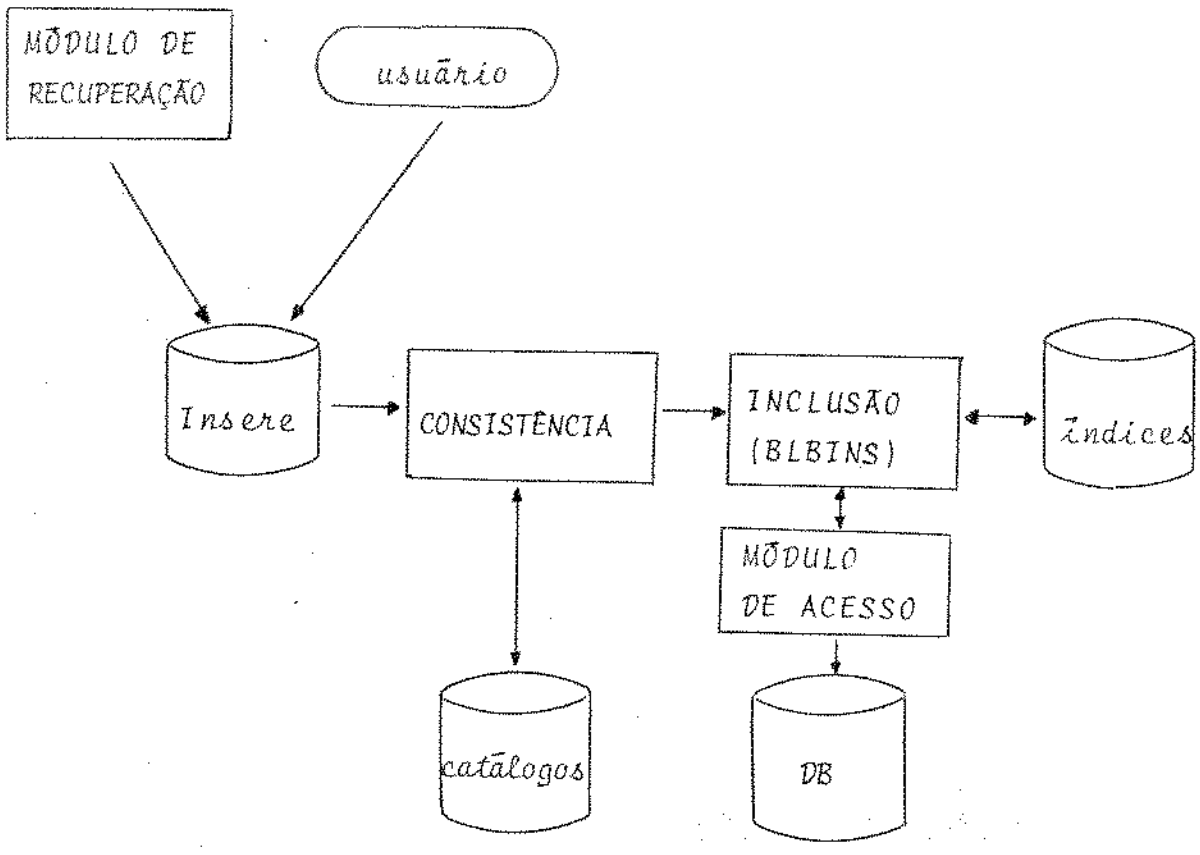


Fig. 3.9 - Processo de Inserção de uma Tupla

Formato: ⌊tid⌋ ⌊rel-id⌋

onde

- tid: identificação interna da tupla a ser removida.
- rel-id: identificação interna da relação onde se encontra a tupla (max. de três caracteres).

A alteração de campos em tuplas será feita através de uma combinação dos dois procedimentos anteriores, ou / seja, o usuário deverá identificar as tuplas e os campos a serem/ alterados através da linguagem SEQUEL no módulo de recuperação.

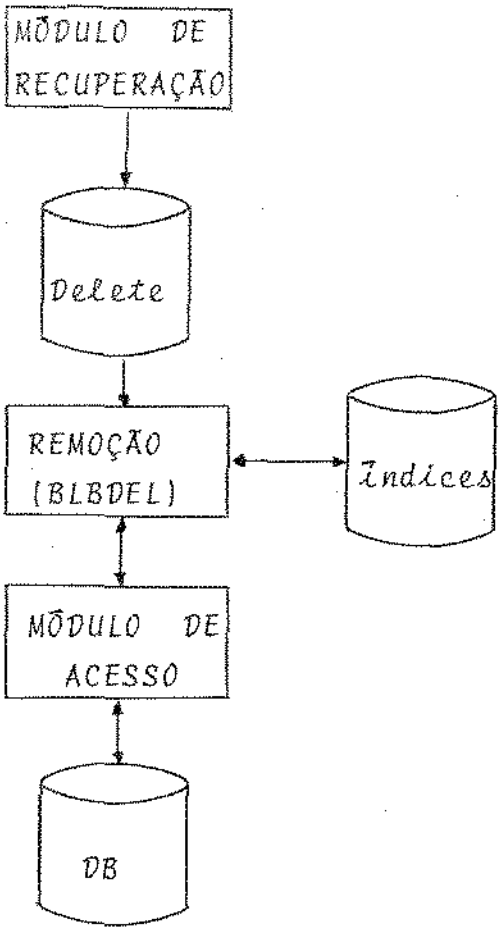


Fig. 3.10 - Processo de Remoção de uma Tupla

Dessa forma, identificadas as tuplas, o sistema pede os novos valores dos campos indicados na clausula SELECT (a linguagem SEQUEL 2 será descrita em 3.7). Para os demais campos é conservado o antigo valor.

Internamente o que ocorre é a colocação da nova versão dessa tupla no arquivo INSERE (ver fig. 3.11).

Durante as inserções e as alterações é verificada a validade dos dados de entrada de acordo com alguns padrões estabelecidos pelo usuário e que estão armazenados num catá

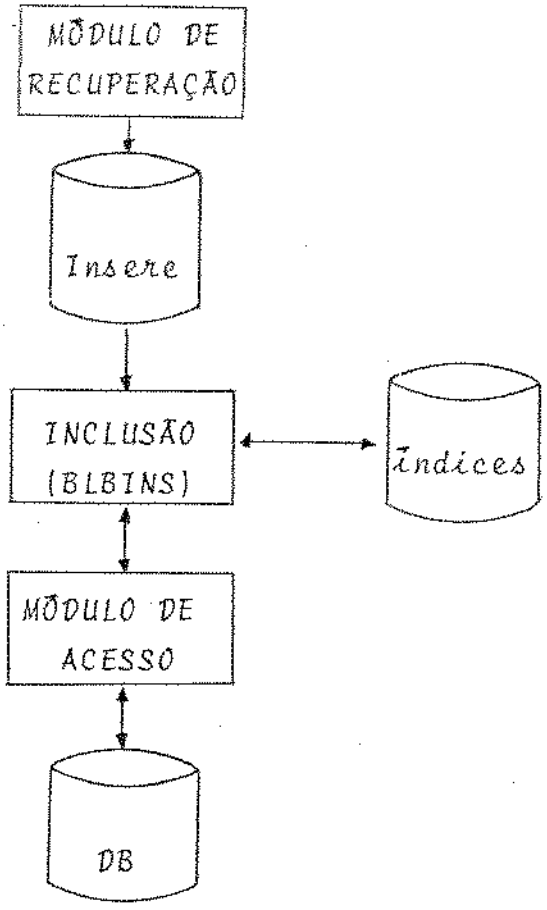


Fig. 3.11 - Processo de Alteração de uma Tupla

logo de características dos dados que compõem as várias tabelas / ou relações do sistema. Esse tipo de controle permite que se fa - çam asserções sobre a integridade dos dados, seu formato, limite/ máximo de número de caracteres se o atributo for de comprimento / variável ou seu comprimento exato se for fixo. Permite-se também/ a especificação de atributos armazenados de uma forma compactada/ em modo binário fixo ou variável que são convertidos internamente pelo sistema.

3.6.2 - MÓDULO DE RECUPERAÇÃO

Para a recuperação dos dados foi inicial - mente implementada uma linguagem simples usando a sintaxe, de expressões booleanas. Mas essa linguagem além das várias limitações sintáticas não era adequada para recuperações envolvendo mais de uma relação.

Decidiu-se então implementar uma linguagem relacional que fosse ao mesmo tempo poderosa em termos de recuperação, mas também fosse fácil de ser utilizada por usuários inexperientes em termos de programação, além de ser do tipo "self-contained", ou seja, ela seria implementada numa sintaxe própria independente da sintaxe de outras linguagens tais como FORTRAN, COBOL, APL, etc...[17] .

Através da vasta literatura existente a / respeito, optou-se pela linguagem SEQUEL, principal interface externa que será suportada pelo Sistema R [23] em desenvolvimento/ no laboratório de pesquisa da IBM em SÃO JOSÉ na California. Esta linguagem opera sobre relações normalizadas como descritas por CODD [12] .

A escolha se deveu à sua simplicidade e à facilidade com que os dados podem ser recuperados, já que não exige como em outras linguagens [13, 14, 15] recursos especiais de máquina para a sua formulação.

Não é objeto desse trabalho a implementa -

ção de todos os recursos de SEQUEL, o que constituiria de per si um projeto de grande porte. Pretendeu-se apenas implementar um subconjunto adequado às finalidades do BIBLOS que será descrito com detalhes na próxima seção.

3.7 - LINGUAGEM DE RECUPERAÇÃO

3.7.1 - INTRODUÇÃO

A definição de um sistema de gerenciamento de banco de dados relacional varia de um que simplesmente suporta um modelo relacional, àquele que implementa totalmente uma linguagem de acesso relacional, tal como a linguagem SEQUEL [24] . No entanto, um aspecto essencial de sistemas relacionais é que os dados são relacionados dinamicamente ao tempo de acesso, ao invés de o serem de uma maneira pré-definida quando foram armazenados.

A fim de se obter essa vantagem com razoável desempenho, o sistema precisa fornecer flexibilidade nos métodos de acesso.

O modelo de dados e a linguagem de dados / associada são as ferramentas básicas para se conseguir a independência dos dados; o modelo de dados é a maneira como o usuário visualiza as informações no banco de dados, e a linguagem dos dados é a linguagem através da qual o usuário transfere informações do modelo para a sua área de trabalho.

Além da introdução da estrutura relacional de dados, Codd [11] definiu dois modelos de linguagem, baseados

em cálculo relacional e algebra relacional, que tem como objetivo servir como base para linguagens de manipulação de dados.

O resultado de qualquer consulta é um conjunto, ou seja, é sempre uma relação derivada,de alguma maneira , das relações que compõem o modelo de dados. De uma maneira geral, o resultado pode ser extraído de uma única relação ou pode envolver duas ou mais relações do modelo de dados; portanto a lingua - gem de consulta deve permitir ao usuário especificar quais as relações que devem ser manipuladas a fim de se conseguir o resultado desejado.

Há pelo menos duas maneiras pelas quais o usuário pode especificar uma consulta:

1 - especificando realmente a sequência de operações de algebra / relacional a serem executadas para produzir o resultado desejado;

2 - simplesmente colocando uma definição do resultado desejado em termos de calculo relacional, deixando que o sistema decida / que operações são necessárias.

Através da algebra relacional, o usuário / pode formular uma consulta, utilizando um procedimento baseado em operadores tais como "join", projeções e restrições [4] , que / tem as relações como operandos (ver apendice D para as definições destes operadores).

Portanto, uma operação em algebra relacional é uma operação que toma uma ou mais relações como operando(s) e produz uma relação como resultado. Linguagens baseadas em algebra relacional tem a vantagem de evitar o uso de quantificadores/

para se expressar as consultas. No entanto, como Codd salientou / [11] , linguagens algébricas exigem que o usuário expresse o co - mando na forma de um procedimento ou sequência de operações; por - tanto requerem uma certa programação por parte do usuário e ao mesmo tempo limitam as oportunidades de otimização dos comandos / por parte do sistema.

O calculo relacional é uma aplicação de / cálculo de predicados. Linguagens de dados baseadas em cálculo re lacional exigem que o usuário especifique uma variável para repre sentar uma tupla de uma relação, e um predicado que defina aque - las tuplas que são de interesse em uma particular consulta. Es - sas linguagens utilizam em seus comandos os quantificadores ∃ (e- xiste) e ∀ (para qualquer).

Com esse tipo de linguagem, uma consulta / pode ser feita sem procedimentos, garantindo a independência dos dados e permitindo que o sistema otimize a execução da consulta.

3.7.2 - LINGUAGEM SEQUEL

Desde a introdução por Codd do modelo de dados relacional, tem sido propostas várias linguagens para manipu lação de banco de dados relacional voltadas para usuários sem ex- periência em programação. Uma dessas linguagens é a SEQUEL.

SEQUEL é uma linguagem de consulta que não está baseada em algebra relacional ou cálculo relacional. É uma linguagem expressa sem procedimentos que não faz uso de quantifi- cadores ou outro conceito matemático qualquer; ela é expressa em

um formato de estrutura de blocos, daí o seu nome "Structured / English Query Language", baseada em palavras-chave da língua inglesa e direcionada para utilização tanto por programadores profissionais como por usuários que não tenham experiência em processamento de dados.

Além da recuperação, a linguagem SEQUEL / possui recursos que permitem inserções, retiradas e atualizações/ de tuplas ou conjunto de tuplas em um banco de dados relacional. Através do recurso de controle de dados, a cada usuário autorizado é permitido autorizar outros usuários no acesso aos dados. Pode-se também com isso estabelecer asserções que garantam a integridade dos dados.

O que determinou a escolha da linguagem SEQUEL como linguagem de consulta do BIBLOS foi a facilidade aparente com que ela pode ser assimilada por usuários comuns [29] / além de dispensar equipamentos ou recursos especiais do sistema / para sua utilização,em contraste com inúmeras outras linguagens / [13, 15] .

O subconjunto adotado no projeto BIBLOS implementa as facilidades de recuperação de dados mas não os comandos de definição e manipulação, já que a prática tem demonstrado, ser mais segura e eficiente a atualização feita "off-line" com a respectiva manutenção dos arquivos invertidos e de controle,particularmente nos casos em que a frequência de atualização é muito / menor que a de consulta.

A linguagem será aqui descrita através de uma série de exemplos baseados no banco de dados da figura 3.12.

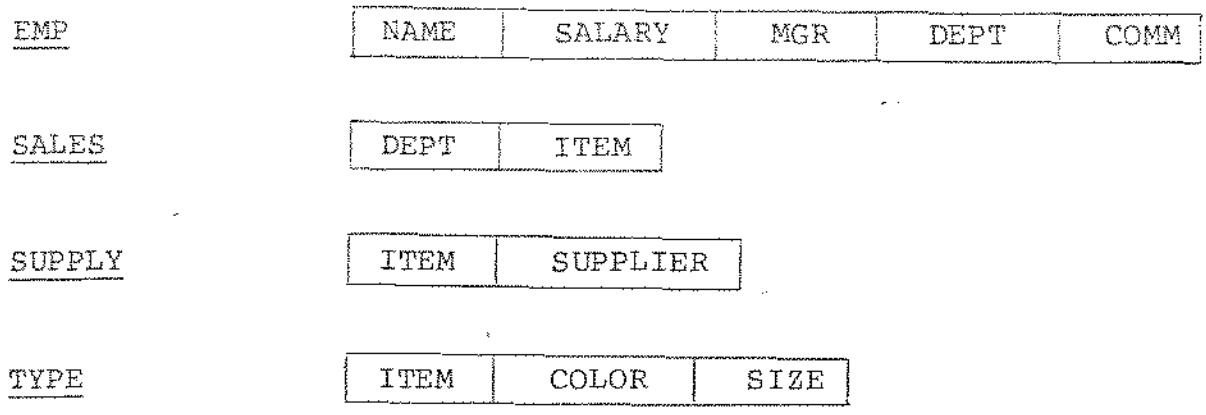


Fig. 3.12 - Banco de Dados utilizado na descrição do SEQUEL2

A relação EMP descreve um conjunto de em - pregados, dando o seu nome, salário, nome do gerente de seu depar tamento, o nome do seu departamento e a sua comissão. A relação / DEPT fornece o nome do departamento e o item vendido por ele. A relação SUPPLY descreve os fornecedores de cada ítem. A relação / TYPE descreve as características dos vários itens. No apendice A está a sintaxe BNF completa da linguagem.

A operação básica da linguagem SEQUEL, cha mada mapeamento é ilustrada no exemplo 1. A clausula SELECT lista os atributos desejados - se toda tupla é desejada pode-se especi- ficar como operando do SELECT simplesmente o nome da relação.

A clausula WHERE especifica uma expressão booleana que compara atributos de tuplas e valores (COLOR = / "BLUE") ou compara atributos entre si (SALARY > COMM).

Ex1: Liste os itens de cor azul.

```
select ITEM
    from TYPE
where COLOR = "BLUE";
```

Em geral, um mapeamento retorna uma cole - ção de valores - os atributos selecionados das tuplas que satisfa zem à clausula WHERE. Valores duplicados não são eliminados do conjunto a menos que o usuário o especifique escrevendo SELECT / UNIQUE. O exemplo seguinte ilustra a projeção da relação TYPE no atributo COLOR.

Ex2: *Liste todas as diferentes cores de itens da relação TYPE*

```
select unique COLOR
from TYPE;
```

Um predicado na clausula WHERE pode testar um atributo em rela - ção a um conjunto como o ilustrado no exemplo seguinte, que mostra também a sintaxe da representação de um conjunto de constantes.

Ex3: *Liste os nomes dos empregados com salário igual a 8000, 12000 ou 9000.*

```
select NAME
    from EMP
    where SALARY isin ⟨8000, 12000, 9000⟩ ;
```

É possível usar o resultado de um mapeamento na clausula WHERE pa ra um outro mapeamento. Essa operação chamada "ninho de mapeamen- tos" é ilustrada no exemplo seguinte. Mapeamentos podem ser ani - nhados até o nível 1Ø na implementação.

Ex4: *Liste os nomes dos empregados que trabalham no departamento/ de vendas do item "DISH".*

```
select NAME
    from EMP
    where DEPT isin
```

```
select DEPT
    from SALES
    where ITEM = "DISH";
```

A linguagem SEQUEL provê funções que podem ser usadas na clausula SELECT como no exemplo 5. As funções são AVG, SUM, COUNT, MAX e MIN. O sistema R permite que um usuário adicione funções ao sistema colocando rotinas em uma biblioteca especial de funções (esse recurso não será fornecido por enquanto, no projeto BIBLOS).

Ex5: Encontre a média dos salários dos empregados do departamento "HOUSEHOLD".

```
select avg (SALARY)
    from EMP
    where DEPT = "HOUSEHOLD";
```

Em geral, duplicatas não são eliminadas do conjunto de valores / qualificados antes da aplicação da função. No entanto, o usuário/ pode explicitamente requerer a eliminação das duplicatas colocando a palavra UNIQUE dentro do argumento da função.

Ex6: Quantos departamentos diferentes estão sob a gerencia de "MURPHY"?

```
select count (unique DEPT)
    from EMP
    where MGR = "MURPHY";
```

Algumas expressões válidas na clausula SELECT são:

```
avg (SALARY) / 52
avg (SALARY) + avg (COMM)
avg (SALARY + COMM)
```

Uma relação pode ser particionada em grupos de acordo com os valo
res de algum atributo e então uma função é aplicada a cada grupo.
Esse tipo de consulta é ilustrada no exemplo 7. A clausula /
GROUPBY é sempre usada junto com uma função. Quando essa clausula
é usada, cada sel-expr precisa ser uma propriedade única do grupo.
Por exemplo, na consulta abaixo, cada grupo de empregados possui/
um único DEPT e uma única média de salários. Se NAME fosse adi -
cionado na clausula SELECT, a consulta resultaria em erro, pois /
NAME não é uma propriedade única de cada grupo.

Ex7: Liste todos os departamentos e a média dos salários de cada um.

```
select DEPT, avg (SALARY)
   from EMP
groupby DEPT;
```

Uma relação pode ser particionada em grupos e então um predicado/
ou conjunto de predicados aplicado para escolher apenas alguns de
les. Esses predicados são sempre baseados em funções e são coloca
dos na clausula HAVING como mostrado no exemplo seguinte:

Ex8: Liste os departamentos nos quais a média dos salários é me - nor que 10000.

```
select DEPT
   from EMP
groupby DEPT
  having avg (SALARY) < 10000;
```

Quando uma consulta possui as clausulas WHERE e HAVING, primeiro/
a clausula WHERE é aplicada para qualificar as tuplas; então os

grupos são formados e a clausula HAVING aplicada para qualificar os grupos como mostrado no exemplo 9.

Ex9: Liste os departamentos sob a gerencia de "SMITH" que empre - gam mais que dois funcionários.

```
select DEPT
    from EMP
where MGR = "SMITH"
groupby DEPT
having count (*) > 2;
```

Note-se o uso do parametro especial "*" (asterisco) para a função COUNT significando uma contagem dos elementos dos grupos formados. Uma função especial chamada SET avalia o conjunto de valores de um dado atributo para todos as tuplas de cada grupo. Este conjunto de valores de atributos pode então ser comparado com um outro/ conjunto como parte da clausula HAVING.

Ex10: Liste os gerentes de departamentos que vendem o item "PEN".

```
select MGR
    from EMP
groupby MGR
having set (DEPT) =
        select DEPT
            from SALES
        where ITEM = "PEN";
```

Os operadores de conjuntos INTERSECT, UNION e SUBTRACT estão também disponíveis. Eles podem ser usados para combinar os resultados de dois mapeamentos como mostra o exemplo 11.

Ex11: Liste os departamentos que vendem itens e que não possuem / empregados.

```
select DEPT
    from SALES
        subtract
select DEPT
    from EMP;
```

Uma consulta pode retornar valores selecionados de mais de uma relação. Um exemplo é a operação "join" na consulta seguinte. O usuário pode listar várias relações na clausula FROM. Conceitualmente, o produto cartesiano dessas relações é formado e seus elementos / são selecionados pelos predicados da clausula WHERE.

Quando mais de uma relação é especificada/ na clausula FROM, o usuário precisa tomar cuidado para qualificar os atributos nas clausulas SELECT e WHERE (exemplo: distinguir / SUPPLY.ITEM de TYPE.ITEM). Quando um nome de atributo ocorre apenas em uma das relações participantes, ele não precisa ser qualificado.

Ex12: Liste os nomes de todos os empregados e os itens vendidos / em seus departamentos.

```
select EMP.NAME, ITEM
    from EMP, SALES
where EMP.DEPT = SALES.DEPT;
```

Em alguns tipos de consultas é necessário juntar uma relação com/ si mesma de acordo com algum critério. Isto pode ser feito listando o nome da relação mais que uma vez na clausula FROM como no /

exemplo 13. Em tais consultas, o usuário pode atribuir um rótulo/ ("label") arbitrário para ser associado a cada uma das relações / participantes. Os rótulos podem ser então usados no lugar dos nomes das relações para quantificar as referencias nas clausulas / SELECT e FROM.

Ex13: Para cada empregado cujo salário exceda o salário de seu gerente, liste o nome do empregado e o nome do gerente.

```
select X.NAME, Y.NAME
   from EMP X, EMP Y
where X.MGR = Y.NAME
   and X.SALARY > Y.SALARY;
```

Uma lista de nomes de atributos entre os caracteres " < " e " > " pode ser usada para denotar uma subtupla de atributos de uma tupla como no exemplo 14.

Ex14: Liste os nomes de empregados do departamento de SMITH que tenham o mesmo salário que ele.

```
select NAME   from EMP
where < DEPT, SALARY > isin
   select DEPT, SALARY    from EMP
   where NAME = "SMITH";
```

Fica assim descrita de maneira sumária a linguagem através de exemplos [24], sem contudo chegar a detalhes de implementação / que serão apresentados nas seções seguintes.

3.7.3 - IMPLEMENTAÇÃO

3.7.3.1 - VISÃO GERAL

Do ponto de vista de arquitetura, o sistema possui uma estrutura de níveis, conforme ilustrado na figura / 3.13. Cada nível é caracterizado por uma linguagem de comunicação, um conjunto de objetos correspondentes a um tipo particular de dado ou modelo sobre o qual a linguagem opera, e um conjunto de funções.

O nível externo se encarrega do que é conhecido na literatura como o esquema conceitual, ou seja, é o nível que suporta a visão lógica dos dados pelo usuário e provê os/ recursos de independência. Neste nível, os objetos são simplesmente conjuntos de relações definidas por seu nome e pelos atributos correspondentes. A comunicação com o usuário é feita através da / linguagem de recuperação.

As funções executadas são principalmente aquelas de verificação e interpretação do comando de entrada; com isso a comunicação com o nível interno é estabelecida relacionando-se as relações e os atributos com seus objetos correspondentes no nível interno. O interpretador toma o comando como uma cadeia de caracteres, analisando-a sintática e semanticamente.

Nesse processo, o comando inicial é convertido em um formato interno inteligível pelo próximo nível.

No nível interno, os argumentos são o nome da relação, os nomes dos atributos e um conjunto de condições a serem satisfeitas. Às relações do nível externo correspondem tabelas com seus respectivos controles e inversões. As funções disponíveis são aquelas que acessam os arquivos invertidos e executam/ as operações disponíveis na linguagem dos comandos de entrada. O

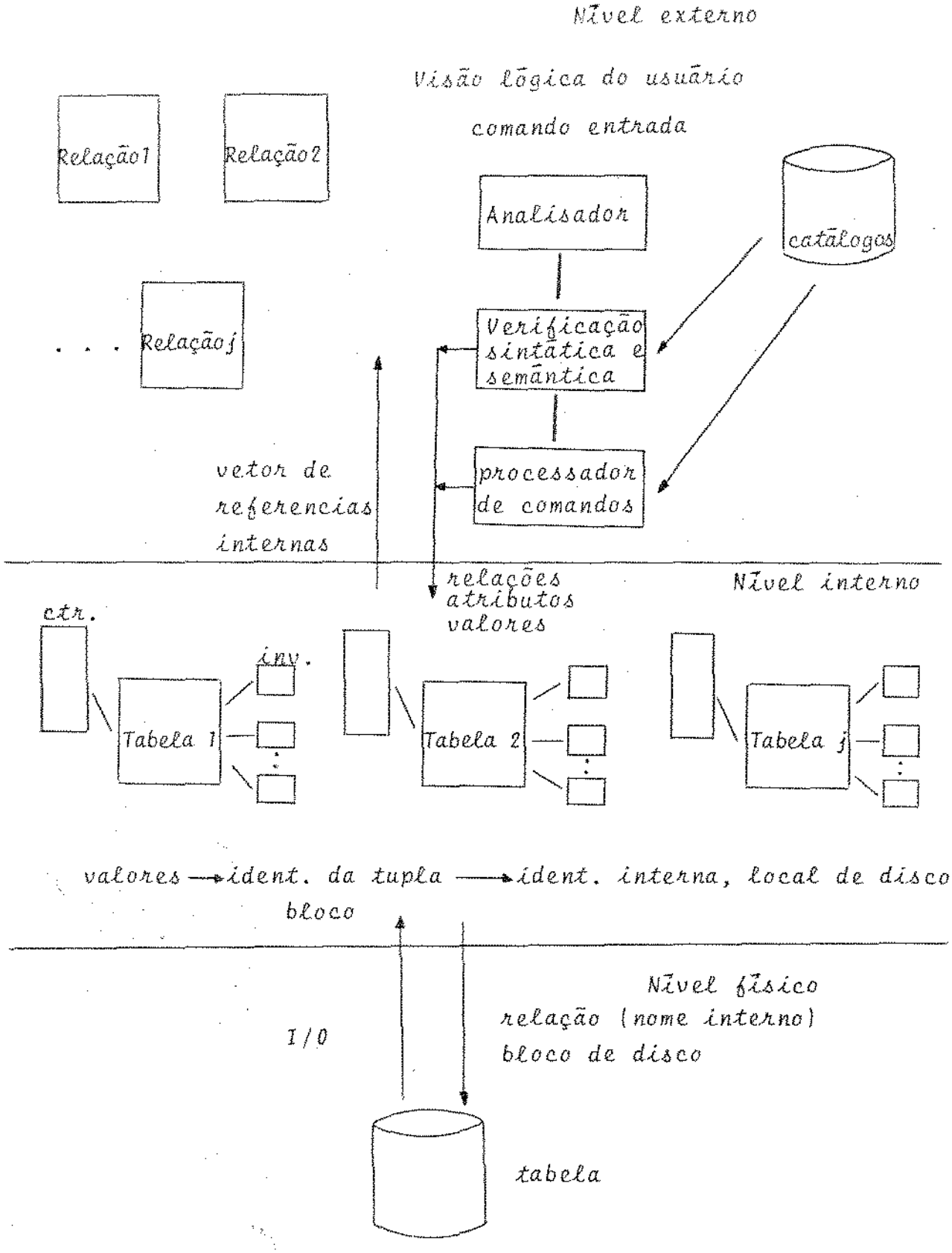


Fig. 3.13 - Arquitetura Geral da Implementação

resultado final é uma seqüência de tuplas que identificam internamente as unidades que satisfazem ao comando de entrada.

O nível físico cuida da transferência do registro físico em disco para a memória através das informações / fornecidas pelo controle de cada tabela.

3.7.3.2 - ANALISADOR SINTÁTICO

Quando se decidiu implementar um subconjunto da linguagem SEQUEL, a idéia era também aproveitar ao máximo / possível todo o software e a estrutura de dados/armazenamento já em uso e suficientemente testados.

Decidido então o subconjunto a ser implementado, a análise sintática ficou a cargo de um programa escrito em PASCAL com analisador do tipo "top-down" recursivo.

Esse programa traduz o comando de entrada/ para uma linguagem intermediária que será manipulada pelo interpretador. Além da análise sintática , as principais funções desse programa são as seguintes:

1) substituir as palavras-chaves da linguagem por códigos que as identificam perfeita e univocamente;
2) marcar o tipo (numérico ou alfanumérico) das constantes que aparecem no comando;
3) colocar as expressões que aparecem no comando no formato / "pos-fixed" que determina a ordem de sua execução;
4) passar os literais para um formato que indica se o literal é de constantes ou de nomes de atributos.

É de se notar que a tarefa do analisador / foi intencionalmente limitada: boa parte do processo de manipulação do comando, especialmente no que se refere à geração de estruturas diretivas do processamento, foi deixada a cargo do interpretador já que este, escrito em FORTRAN-1Ø, permite uma maior facilidade quanto à manipulação de bytes de qualquer tamanho e de arquivos de acesso direto através de chamadas a rotinas escritas em linguagem de montagem (MACRO-1Ø).

3.7.3.3 - ANÁLISE SEMANTICA DOS COMANDOS DE CONSULTA

Os comandos da linguagem SEQUEL, para efeito de implementação dos módulos de análise sintática, semantica e interpretação foram analisados como sendo uma composição de / query-block's conectados pelos operadores de conjuntos INTERSECT, UNION e MINUS podendo o resultado ser classificado pelo aparecimento da clausula ORDER BY (que na primeira versão não será implementada).

Cada query-block retorna dados contidos numa estrutura que pode ser usada como operando por um query-block/ envolvente.

Cada query-block é caracterizado pelos tipos de operações que contêm, distinguindo-se os seguintes tipos:

1 - que acessam uma única tabela ou relação:-

select-clause FROM from-list

[WHERE boolean]

onde from-list especifica uma única relação.

2 - que acessam a mais de uma relação

select-clause FROM from-list

WHERE boolean

onde from-list especifica mais de uma relação.

3 - que acessam uma única relação e possuem a clausula GROUPBY

select-clause FROM from-list

[WHERE boolean]

GROUPBY field-spec [HAVING boolean]

onde from-list especifica somente uma relação.

Vale ressaltar aqui que esta caracteriza - ção só visou facilitar a implementação da linguagem. Isto porque/ sua sintaxe geral produz construções que não são válidas semanticamente e a classificação proposta acima permite detetar tais inconsistencias.

Transcreve-se a seguir a sintaxe do subconjunto de SEQUEL que foi implementado, indicando-se entre parenteses os pontos de possível inconsistencia para as construções dos diferentes tipos.

- query :: = query-expr;
- query-expr:: = query-block /

query-expr set-op query-block /

(query-expr)

- set-op :: = INTERSECT / UNION / SUBTRACT
- query-block :: = select-clause FROM from-list (1)

[WHERE boolean]

[GROUPBY field-spec [HAVING boolean]](2)

(1) Em query-block's do tipo três pode aparecer uma só relação na

clausula FROM.

(2) Para que essa construção fosse permitida em query-block's do tipo dois teríamos que ou efetivar o produto cartesiano e depois agrupar os elementos segundo algum critério, ou alternativamente, após simulado o produto cartesiano, agrupá-los pelo "field-spec" determinado pela clausula GROUPBY. Mas em / qualquer uma das opções, o processo de implementação se torna mais complexo. No primeiro caso a quantidade de memória necessária para se guardar todo o produto cartesiano é impraticável, fora o espaço necessário para se guardar os elementos / dos conjuntos formados.

Por outro lado quando se simula a execução do produto cartesiano, o sistema usa internamente "buffers" de memória para guardar os resultados intermediários, à medida que os registros vão sendo selecionados pela expressão da clausula WHERE; persistindo pois o problema de espaço em memória.

Devido a tais dificuldades, embora não constitua propriamente uma inconsistência semântica, deixar-se-á de implementar na primeira versão este tipo de comando, especialmente em se considerando que sua utilidade é bastante questionável.

- select-clause :: = SELECT [UNIQUE] sel-expr-list

- sel-expr-list :: = sel-expr / sel-expr-list, sel-expr (3)

(3) Em query-block's do tipo três, as expressões aritméticas da clausula SELECT bem como as que aparecem na clausula HAVING / devem se referir a valores comuns aos grupos que estão manipu

lando. Ou seja, só é permitido o aparecimento de funções ou o nome do atributo pelo qual foram formados os grupos.

- sel-expr :: = expr / var-name / table-name (4)

(4) Essa construção em query-block's do tipo um permite que se recupere dos registros selecionados pela expressão booleana , todos os atributos da relação.

- from-list :: = table-name [var-name] /
  from-list, table-name [var-name] (5)

(5) Em comandos que acessam mais de uma relação, o produto cartesiano dessas relações é conceitualmente formado e então seus/ elementos selecionados pelos predicados da clausula WHERE.

- field-spec-list :: = field-spec / feld-spec-list, field-spec
- boolean :: = boolean-term / boolean OR boolean-term
- boolean-term :: = boolean-factor /
  boolean-term AND boolean-factor
- boolean-factor :: [NOT] boolean-primary
- boolean-primary :: = predicate / (boolean)
- predicate :: = expr comparison expr /
  expr BETWEEN expr AND expr /
  expr comparison table-spec /
  (6) ⟨field-spec-list⟩ = table-spec /
  (6) ⟨field-spec-list⟩ [IS][NOT] IN table-spec /
  (7) SET (field-spec-list) comparison table-spec /
  (7) SET (field-spec-list) comparison SET (field-spec list)

(6) Não podem aparecer na expressão da clausula HAVING em query - blocks do tipo três.

(7) Não é permitido esse tipo de construção em query-block's do tipo um e dois já que a função SET opera sobre um conjunto de registros, no caso aquele formado pela clausula GROUPBY.

- table-spec :: = query-block / (query-expr) / literal

- expr :: = arith-term / expr add-op arith-term

- arith-term :: = arith-factor / arith-term mult-op arith-factor

- arith-factor :: = [add-op] primary

- primary :: = field-spec / set-fn ([UNIQUE]) expr) /
(8) COUNT (*) / constant / (expr)

(8) A chamada desta função com este parâmetro retorna o número de elementos dos conjuntos formados pela clausula GROUPBY.

- field-spec :: = field-name / table-name. field-name /
var-name. field-name

- comparison :: = [NOT] CONTAINS / IS [NOT] IN / comp-op

- comp-op :: = = / ## / < / > / <= / =>

- add-op :: = + / -

- mult-op :: = * / /

- set-fn :: = AVG / MAX / MIN / SUM / COUNT

- literal :: = (entry-list) / constant

- entry-list :: = entry / entry-list, entry

- entry :: = constant

- constant :: = quoted-string / number / NULL

- table-name :: = identifier

- field-name :: = identifier

- var-name :: = identifier

3.7.3.4 - INTERPRETADOR

O interpretador foi implementado em dois módulos visando facilitar uma possível otimização da execução dos comandos. O primeiro módulo, o pré-processador, substitui todos os nomes externos de relações e atributos pelas identificaçoes internas correspondentes. Nessa tradução são utilizados os catálogos de conversão do sistema. Paralelamente é criada uma tabela das variáveis especificadas na clausula FROM que são substituídas pelas identificações internas das respectivas relações, para a execução do comando.

Feita a traduçao, o pré-processador cria uma outra tabela que determina a ordem em que os "query-block's" devem ser avaliados. Isso se deve ao fato de que um "query-block" mais interno deve ser resolvido antes que outro mais externo, já que o seu resultado será utilizado como operando.

Em seguida, o código intermediário gerado e esta última tabela são passados como parâmetros para o segundo

módulo: o processador propriamente dito, que conforme o tipo do "query-block" (vide seção 3.7.3.3) executa-o apropriadamente.

Deve-se ressaltar que para uma maior otimização na interpretação dos "query-block's", pode-se em versões futuras, incluir uma série de parâmetros adicionais para o processador; por exemplo, que estruturas (índices, listas, etc...) estão disponíveis e poderiam ser utilizadas para se aumentar a eficiência de processamento.

As figuras 3.14, 3.15 e 3.16 mostram como são interpretados os três tipos de "query-block's", apresentando/ o fluxograma dos respectivos algoritmos.

Para a manipulação de "joins", BIBLOS usa o método de "loops aninhados". Em "joins" envolvendo duas relações, uma delas é chamada de relação externa, de onde uma tupla é primeiramente extraída, e a outra é a relação interna de onde tuplas serão possivelmente recuperadas, dependendo dos valores obtidos na tupla da relação externa.

Um predicado que relaciona atributos de / duas relações a serem "joined" é chamado de predicado "join".

O método de "loops aninhados" consiste em se percorrer a relação externa e, para cada tupla, fazer o seguinte: recuperar uma de cada vez, as tuplas da relação interna que / satisfazem o "join". As tuplas compostas, formadas pelos pares de tuplas da relação externa-tuplas da relação interna constituem o resultado do "join".

Sobre esse conjunto é então aplicada a clausula SELECT do "query-block" correspondente.

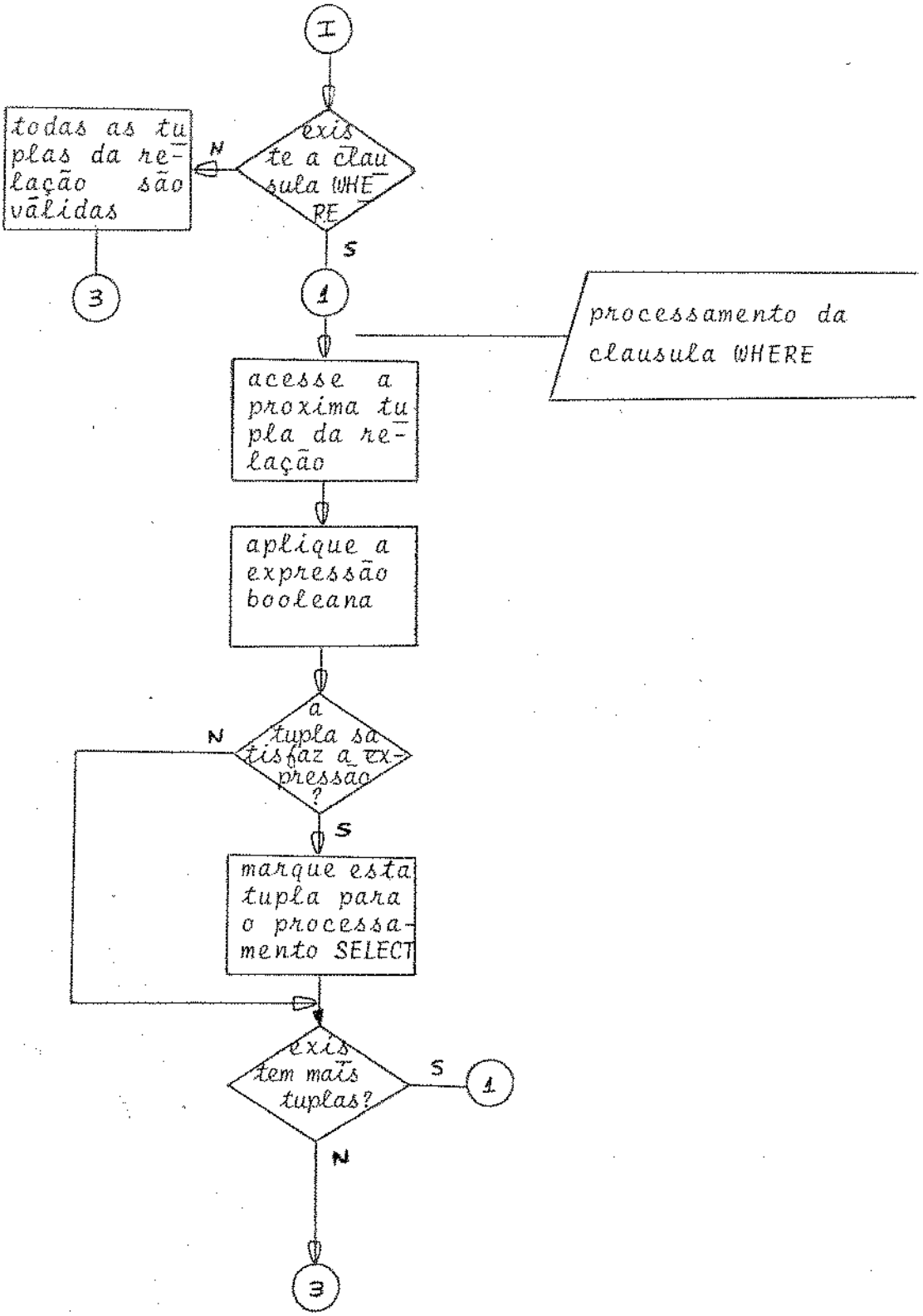


Fig. 3.14 - Interpretação de query-block's do tipo 1

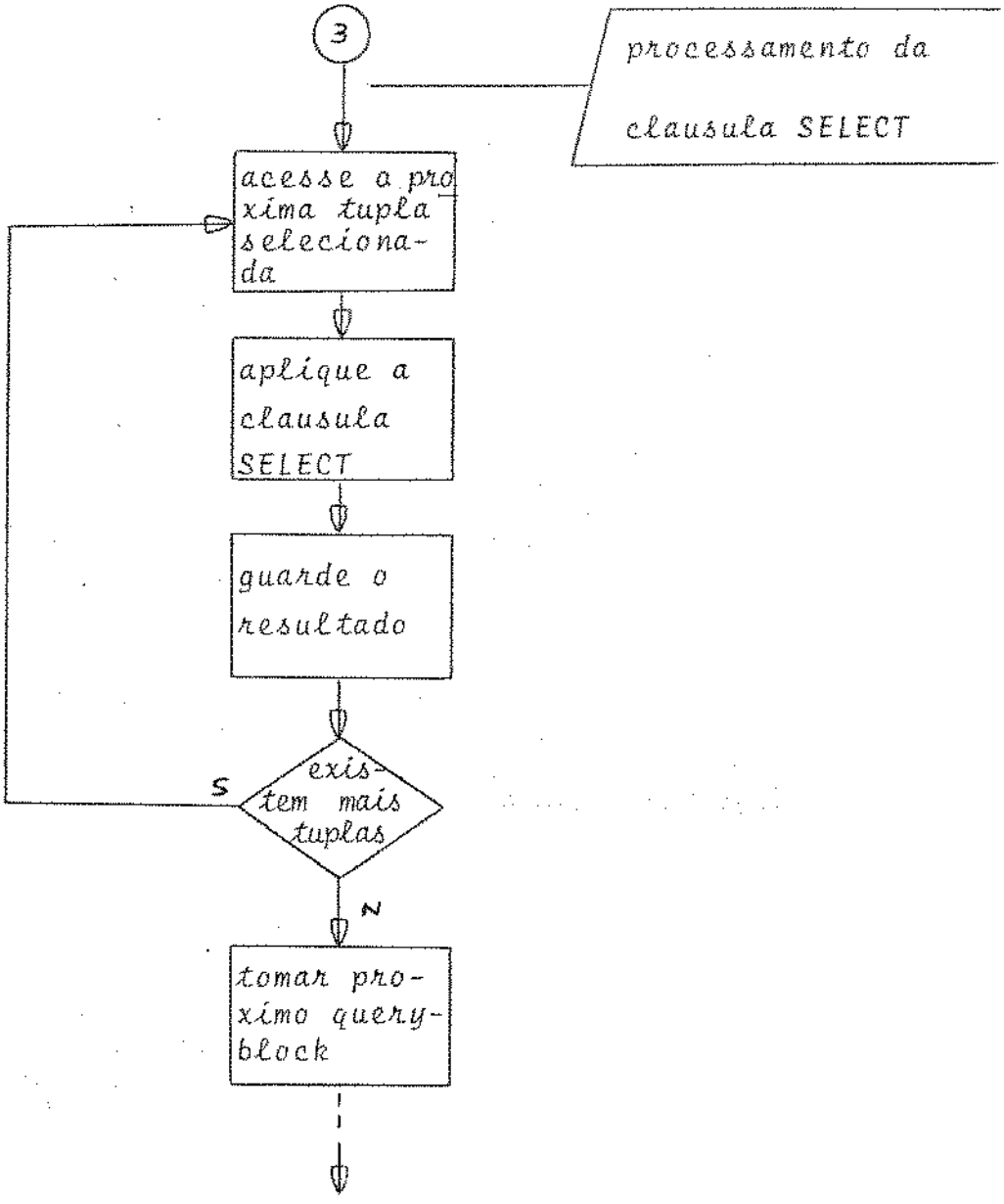


Fig. 3.14 - (Continuação)

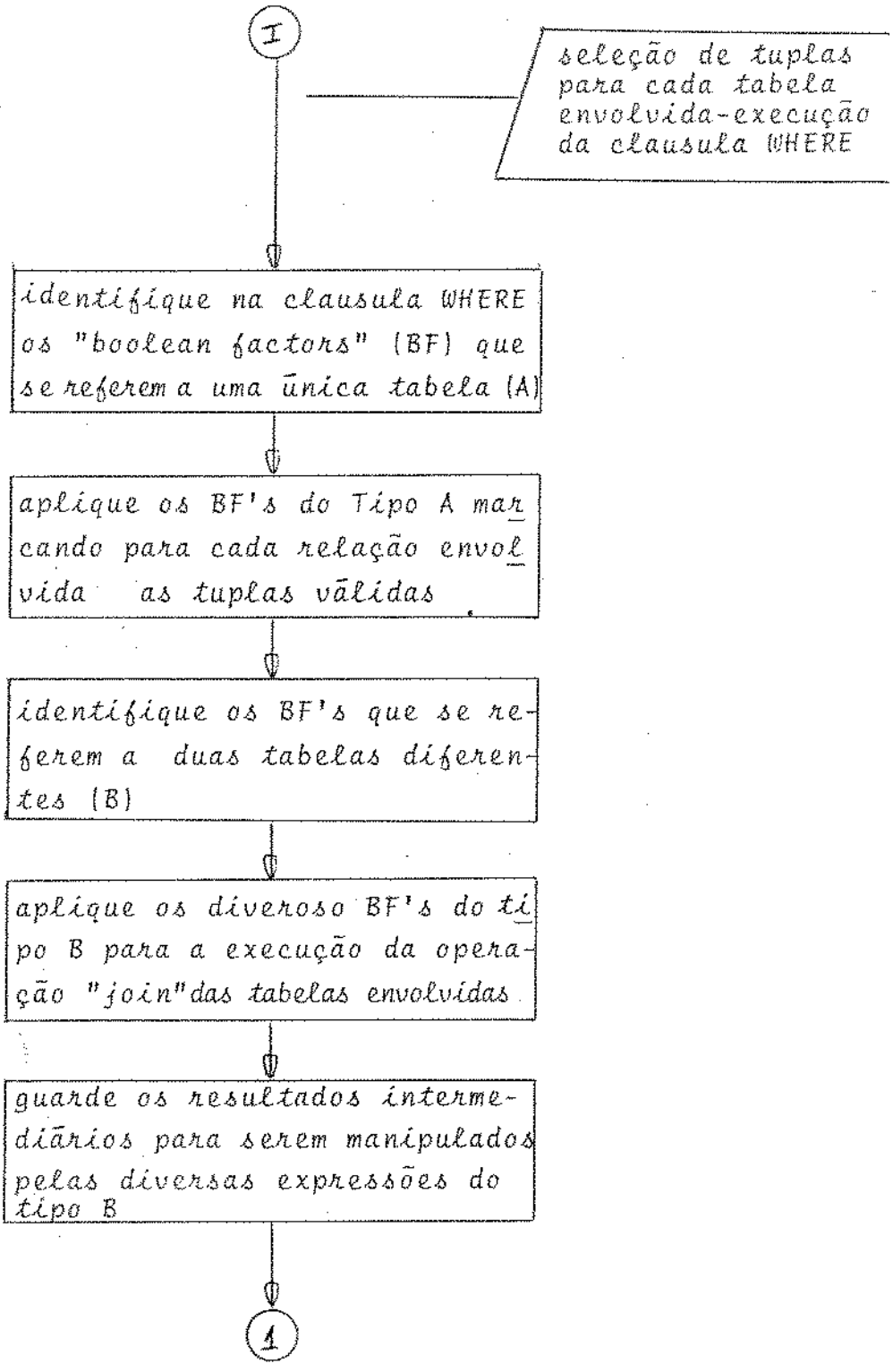


Fig. 3.15 - Interpretação de query-blocks do tipo 2

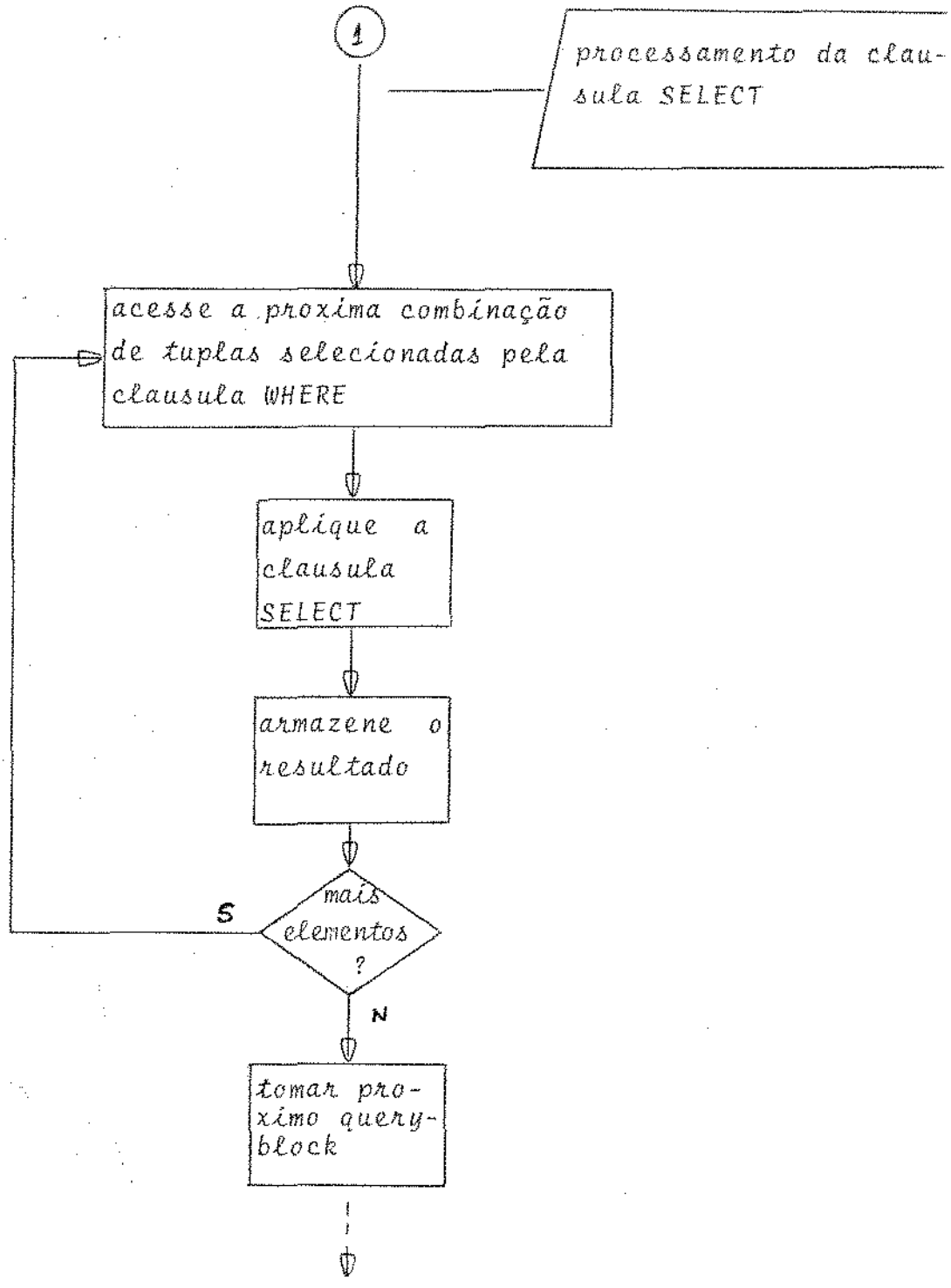


Fig. 3.15 - (Continuação)

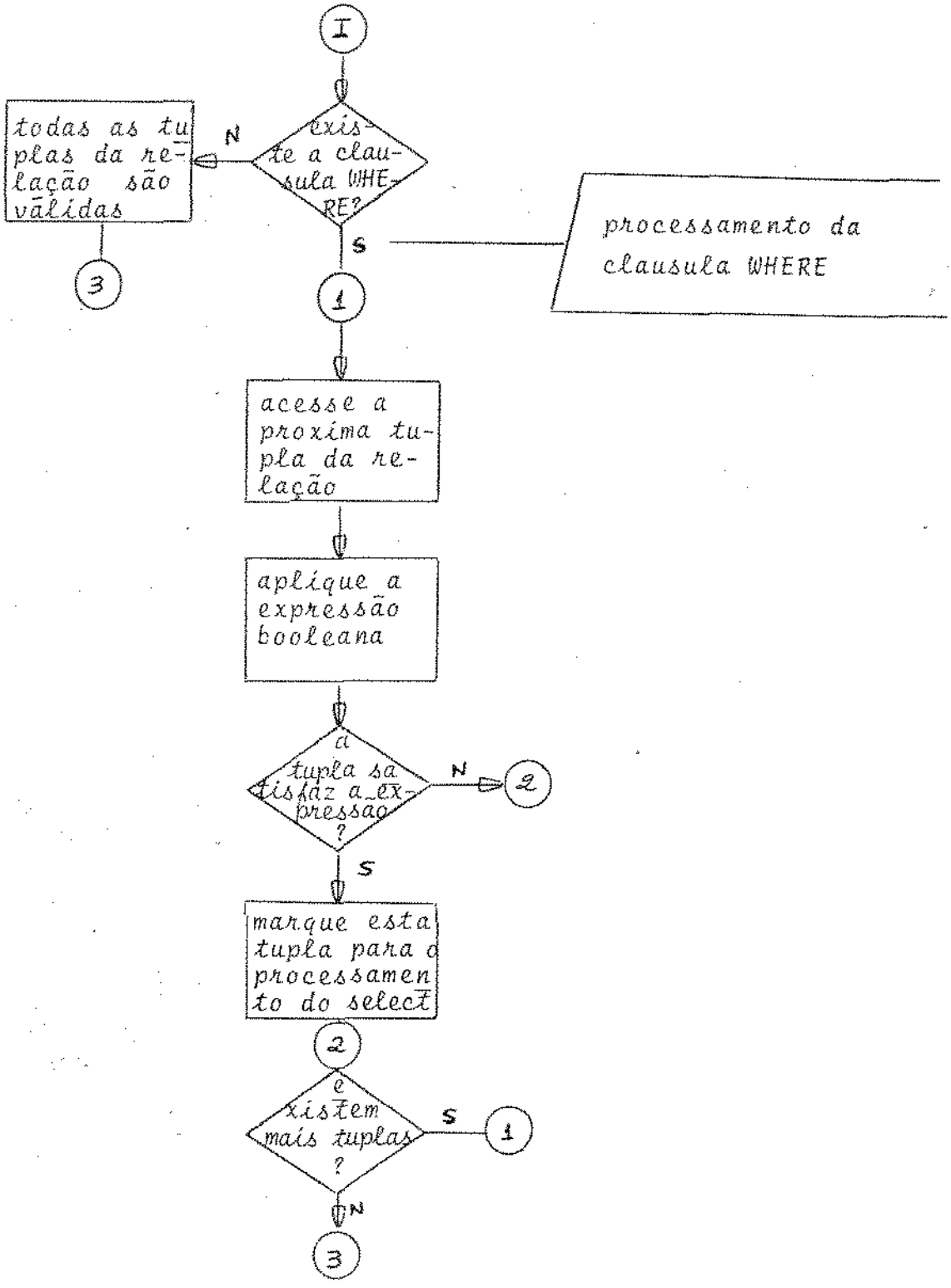


Fig. 3.16 - Interpretação de query-block do tipo 3

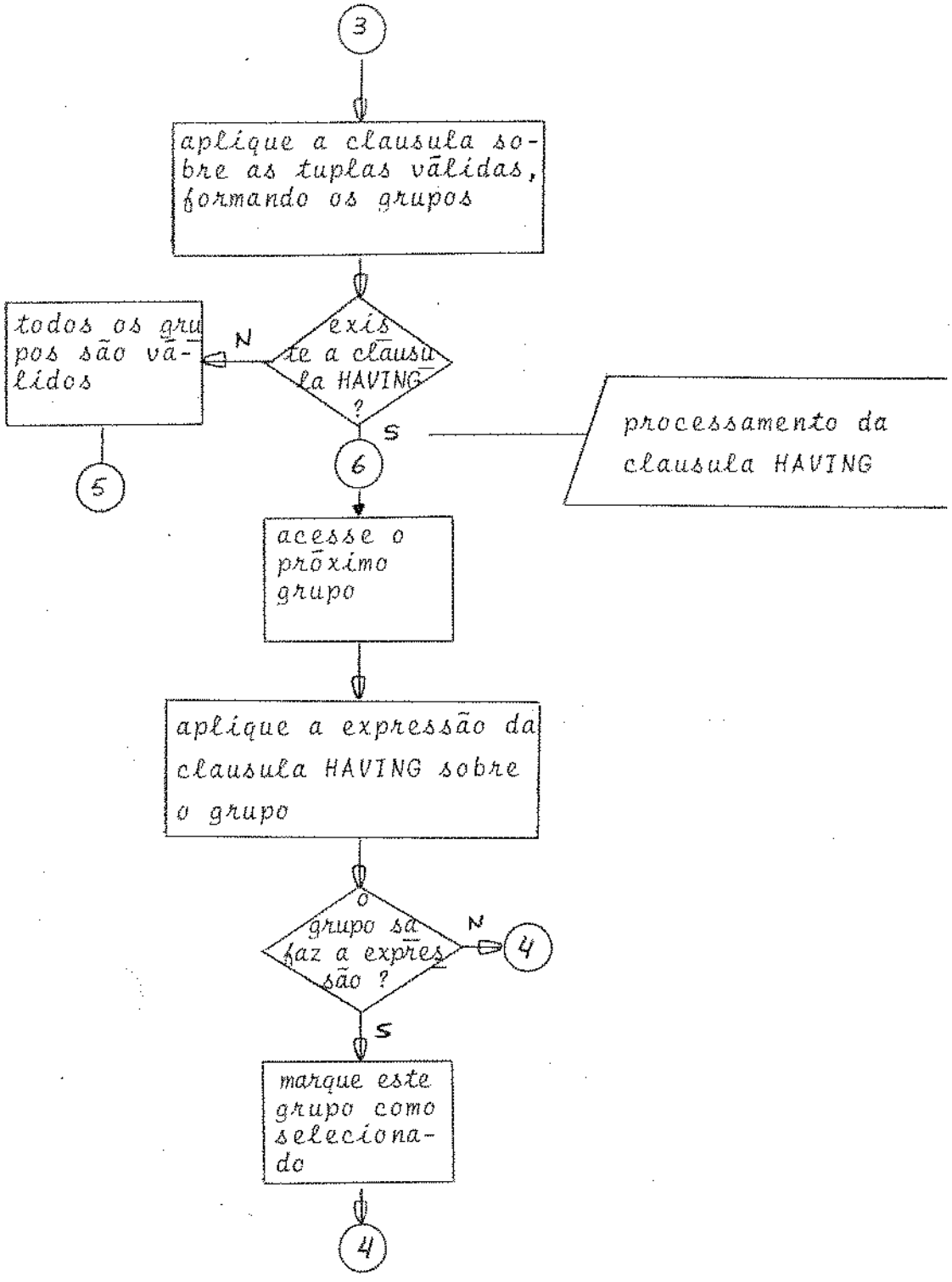


Fig. 3.16 - (Continuação)

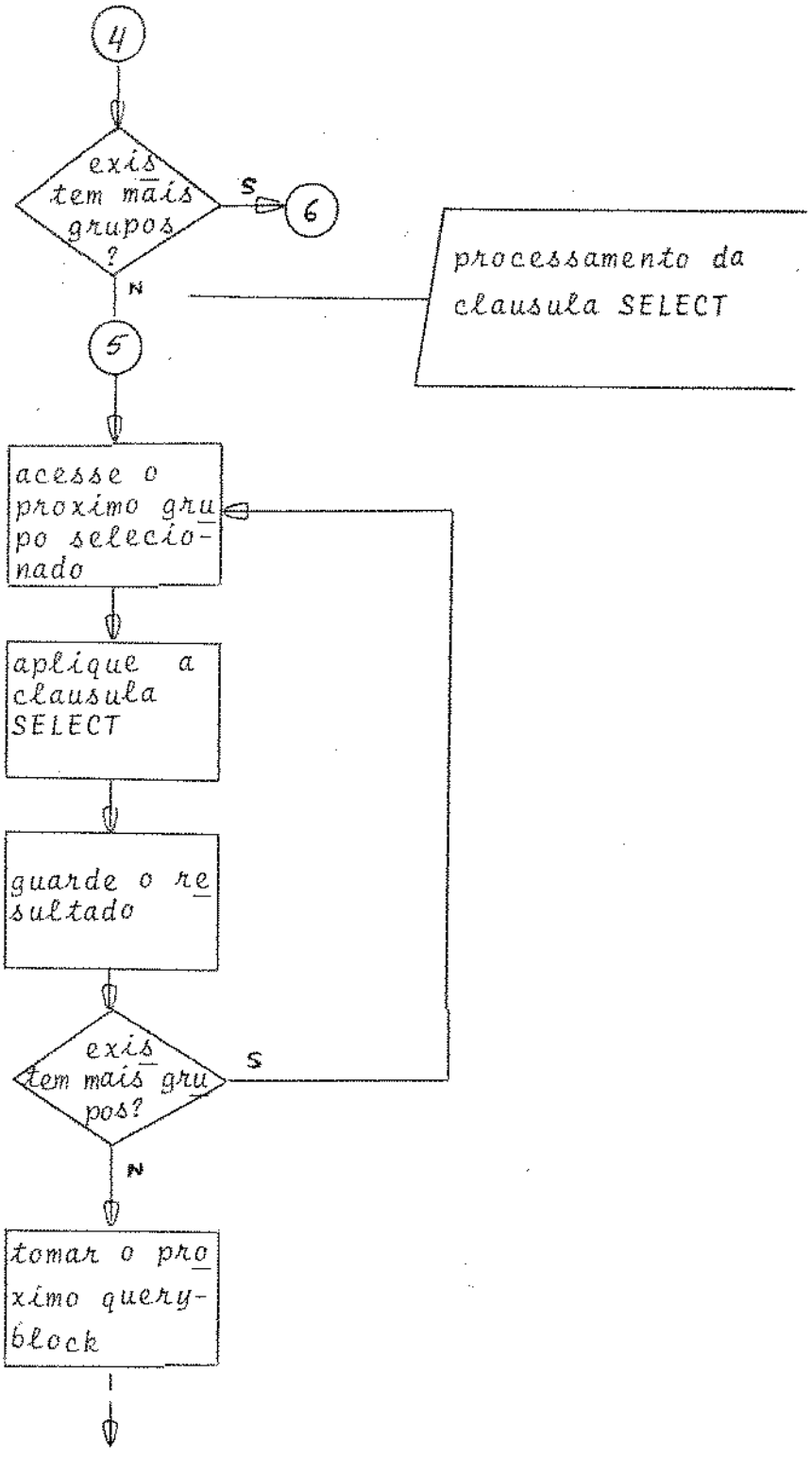


Fig. 3.16 - (Continuação)

4 - CONCLUSÕES FINAIS E SUGESTÕES PARA DESENVOLVIMENTO FUTURO

Desde a sua introdução por Codd [4], o modelo relacional tem sido estudado com aplicações teóricas em laboratórios de pesquisa, sendo ainda raros atualmente os sistemas de gerenciamento de banco de dados puramente relacionais disponíveis para venda no mercado. Tal não acontece com os outros dois modelos, o hierárquico e o de redes, largamente difundidos.

Dessa forma, são muito pouco conhecidas as características práticas do modelo relacional, a menos as do Sistema R, sobre o qual se encontra disponível uma farta bibliografia. [24], [27], [30], [31], [32], etc...

Como havia a necessidade de se expandir o sistema BIBLOS original, decidiu-se optar pela sua ampliação utilizando o modelo de dados relacional tornando-o, em princípio, capaz de atender à demanda prevista na área administrativa da UNICAMP.

A característica relacional do sistema desenvolvido foi ilustrada através da utilização, como amostra de dados, das quatro relações expostas em 3.7.2. A escolha dessa amostra visou principalmente a utilização de dados que comumente aparecem como exemplo na literatura, para se proceder a uma análise comparativa de capacidade, eficiencia e custo.

A escolha do modelo relacional, forçou a implementação de uma interface apropriada entre o sistema e o usuário, de forma a permitir que este último utilize as informações

armazenadas, sem precisar ter conhecimento da estrutura interna de armazenamento e dos métodos de acesso disponíveis.

A interface escolhida foi a linguagem SE - QUEL 2, baseando-se novamente no sistema R. Esta linguagem é de fácil assimilação e muito poderosa em termos de recuperação, definição e manipulação de dados (vide apendice A).

As características das aplicações pretendidas indicaram a conveniencia de se implementar inicialmente apenas um subconjunto dos recursos do SEQUEL 2, ou seja, os de recuperação; os de definição e manipulação de informações foram substituídos por procedimentos "off-line" já implementados na versão anterior do BIBLOS.

Na sua versão atual, BIBLOS terá que sofrer grandes modificações para atender a aplicações que dispunham de maior quantidade de informações armazenadas. As novas técnicas a serem empregadas terão que resolver principalmente problemas de otimização de memória, para diminuir os acessos a disco.

Por esse motivo, o sistema nesta versão deve ser considerado pelas suas intenções e potencialidades, como um protótipo de um sistema de gerenciamento de banco de dados relacional mais completo. Com a implementação das sugestões expostas abaixo pretende-se expandi-lo de forma a permitir o seu uso eficiente em outras aplicações maiores.

O tempo de resposta às consultas formuladas (apendice C) ainda não satisfaz plenamente as expectativas, mas devemos considerar que o processamento é feito num sistema "timesharing" concorrendo, em média, com pelo menos outros trinta

usuários do DEC-10 da UNICAMP.

Uma limitação muito séria neste tipo de projeto é a quantidade de memória exigida para o processamento eficiente de um comando da linguagem SEQUEL. Para se chegar a um equilíbrio entre os recursos disponíveis e a eficiência de processamento é necessário utilizar técnicas cada vez mais sofisticadas.

Um outro fato a ser ressaltado é a necessidade de se conciliar, em implementações reais, o modelo relacional teórico com as necessidades práticas de eficiência em termos de tempo de CPU e utilização de memória. Para essa adaptação empregam-se uma série de estruturas de dados (determinadas a priori pelo DBA) que não são previstas no modelo teórico, tais como índices, apontadores, "clusters", etc... que constituem na verdade uma maneira eficiente de se atender às consultas feitas pelos usuários. É importante frisar que tais estruturas "satélites" não são intrinsicamente necessárias para se acessar o banco de dados, já que, em princípio, as tuplas podem ser acessadas sequencialmente.

Fica como sugestão para trabalhos futuros a implementação dos seguintes ítens:

1 - Clausula ORDER BY da linguagem SEQUEL;

2 - Variáveis definidas em uma clausula SELECT e usadas em outra mais interna:

Ex: Liste os fornecedores que fornecem todas as partes usadas pelo departamento 50:

```
select SUPPLIER
from SUPPLY X
where
```

```
        (select PART
         from SUPPLY
         where SUPPLIER = X. SUPPLIER)
    contains
        (select PART
         from USAGE
         where DNO = 50)
```

3 - Permitir que o usuário crie suas próprias funções e as guarde numa biblioteca de onde serão acessadas pelo processador da linguagem SEQUEL;

4 - Permitir que funções sejam chamadas recursivamente;

5 - Permitir que o usuário possa armazenar resultados de consultas como novas relações ("visões") que podem por sua vez, serem acessadas normalmente por outros comandos;

6 - Implementação de todo o conjunto dos números reais;

7 - Permitir que a clausula GROUPBY seja aplicada a atributos não indexados;

8 - Implementação de catálogos de proteção, ou seja, proteção a nível de campos de informação e relações seja para consultas ou alterações;

9 - Desenvolver as técnicas de processamento da expressão booleana na clausula WHERE nos tres tipos de "query-blocks", utilizando as estruturas de informação (índices, apontadores , etc...) disponíveis. [26]

BIBLIOGRAFIA

/1/ MCGEE,W.C , "Generalization: Key to Succesfull Electronic Data Processing" - JACM 6,1 (jan 59), pp. 1-23.

/2/ MCGEE,W.C. & TELLIER,H., "A Re-evaluation of Generalization" Datamation, (july-august 1960), pp 25-38.

/3/ EVEREST,G.C. , "The Objectives of Data-base Management" - Information Systems COINS IV (Tou) Plenum Press, New York - 1974, pp 1-35.

/4/ CODD,E.F. , "A Relational Model of Data for Large Shared Data Banks" - IBM Research Laboratory, San Jose,California , CACM - june,1970, vol. 13, no. 6.

/5/ DATE,C.J. , "An Introduction to Database Systems" - IBM (UK) Laboratories Ltda - Addison-Wesley Reading - 1975, 366 pp.

/6/ SEVERANCE,D.G. & CARLIS,J.V. , "A Practical Approach to Selecting Record Access Paths" - Computing Surveys, vol 9 , no. 4, december 1977.

/7/ KNUTH,D.E. , "The Art of Computer Programming" - vol. 1: Fundamental Algorithms, Addison-Wesley Reading - 1968, 634 pp.

/8/ BUCHOLZ,W. , "File Organization and Addressing" - IBM S.J. junho 1958, pp 80-111.

/9/ TURNER,M.J.;Hammond R. & COTTON,P. - "A DBMS for Large Statistical Databases" - VLDB - outubro 1979, pp 319-325.

/10/ BABAD,J.M. , "A Record and File Partitioning Model" - Uni - versaty of Chicago , CAMC - jan,1977, vol 20, no. 1.

/11/ CODD,E.F. , "Relational Completeness of Data Base Sublangua- ges" - In Courant Computer Science Symposia, vol 6: Data Base Systems, G.Forsythe,Ed. Prentice Hall, Engelwood Cliffs, N.J., 1971, pp 65-98.

/12/ CODD,E.F. , "Normalized Data Base Structure: A Brief Tutori- al" - Proc. 1971 , ACM SIGFIDET - Workshop on Data Des- cription, Acess and Control.

/13/ ZLOOF,M.M. , "Query-Example" - IBM Systems Journal - no. 4 1977.

/14/ ANTONACCI,F.; DELL'ORCO,P.; SPADAVECCHIA,V.N et all , " AQL- A Problem-solving Query Language for Relational Data Bases" IBM J. R. Development , vol 22, no. 5, sep. 1978.

/15/ BOYCE,R.F.; CHAMBERLAIN,D.D. et all , "Specifying Queries as Relational Expressions, The Square Data Sublanguage" - IBM Research Laboratory, San Jose - CACM - nov. 1975, vol. 18, no. 11.

/16/ EARLEY,J. , "On the Semantics of Data Structures" - Universi ty of California, Berkeley - In Courant Computer Science Symposia, vol. 6:Data Base Systems, G.Forsythe,Ed.Prentice Hall, Engelwood Cliffs, N.J., 1971, pp 23-32.

/17/ DBMS-10 - Data Base Management System : Administrator's Manu al , jun 1977 (AA0899C-TB)

/18/ BLASGEN,M.W. & ESWARAN,K.P. - "Storage and Access in Relatio

nal Data Bases" - IBM Systems Journal , no. 4, 1977.

/19/ UHROWCZIK,P.P. , "Data Dictionary/Diretories" - IBM Systems journal - no. 4, 1973

/20/ ADABAS , User Manual - order number: ADA 410-000.

/21/ CODD,E.F. , "Further Normalization of the Data Base Relational Model" - In Courant Computer Science Symposia, vol. 6 Data Base Systems, G.Forsythe,Ed. Prentice Hall, Engelwood Cliffs, N.J., 1971, pp 33-64.

/22/ CHAMBERLAIN,D.D. , "Relational Data Base Management Systems" Computing Surveys , vol. 8, no. 1, mar 1976.

/23/ ASTRAHAN,M.M.; BLASGEN,M.W. et all , "Systema R: Relational Approach to Database Management" - ACM Transactions on Database Systems, vol. 1, no. 2, june 1976, pp 97-137.

/24/ CHAMBERLAIN,D.D.; ASTRAHAN,M.M et all , "SEQUEL 2:A Unified Approach to Data Definition, Manipulation and Control" - IBM J.Res.Development , nov 1976, pp 560-575.

/25/ STONEBRAKER,M.;WONG,E. et all , "The Design and Implementation of INGRES" - ACM Transactions on Database Systems, / no. 3, set 1976, pp 189-222.

/26/ DELOBEL,C.;CASEY,R.G. , "Decomposition of a Data Base and the Theory of Boolean Switching Functions" - IBM J.Res. Development, sept 1973.

/27/ ASTRAHAN,M.M. & CHAMBERLAIN,D.D. , "Implementation of a Structure English Query Language" - IBM Research Division San Jose,CACM , oct 1975, vol. 18, no. 10.

/28/ LANDES;O.E. & MENASCÉ,D.A. , "Um Estudo sobre Tēcnicas de Re

cuperação de Erros em Bancos de Dados" - RBC, Rio de Janeiro , vol. 1, no. 1, 1981.

/29/ REISNER,P. , "Human Factors Studies of Database Query Languages: A Survey and Assesment" - IBM Research Laboratory RJ 3070 (38116) 3/2/81, San jose, California, 95192.

/30/ KIM, WON, "On Optimizing an SQL-LIKE Nested Query" - IBM Research Report - RJ 3063 (37958), 2/23/81

/31/ BLASGEN, M.W, ASTRAHAN, M.M et al,"System R: An architectural Update" - IBM Research Report - RJ 2581 (33481), 7/17/79

/32/ CHAMBERLIN, D.D., Astrahan, M.M et al, "Support for Repetitive Transactions and ad-hoc Query in System R" - IBM Researd Report - RJ 2551 (33151) - 5/22/79

A P Ê N D I C E S

APÊNDICE A

SINTAXE BNF COMPLETA DA LINGUAGEM SEQUEL 2

```
statement :: = query
             | dml-statement
             | ddl-Statement
             | control-statement
             | cursor-statement
dml-statement :: = assignment
                 | insertion
                 | deletion
                 | update
query :: = query-expr [ORDER BY ord-spec-list]
assignment :: = ASSIGN TO receiver : query-expr
              | ASSIGN TO receiver : CURSOR cursor-name

receiver :: = table-name [( field-name-list)]
field-name-list :: = field-name
                   | field-name-list, field-name
insertion :: = INSERT INTO receiver : insert-spec
insert-spec :: = query-expr
               | lit-tuple
deletion :: = DELETE table-name [var-name]
                                [where-clause]
update :: = UPDATE table-name [var-name]
              SET set-clause-list [where-clause]
where-clause :: = WHERE boolean
                | WHERE CURRENT OF cursor-name
set-clause-list :: = set-clause
                   | set-clause-list, set-clause
```

```
set-clause :: = field-name = expr
              | field-name = (query-block)
query-expr :: = query-block
              | query-expr set-op query-block
              | (query-expr)
set-op :: = INTERSECT | UNION | MINUS
query-block :: = select-clause [INTO target-list]
                          FROM from-list
                           [WHERE boolean ]
                           [GROUP BY field-spec-list
                           [HAVING boolean ] ]
select-clause :: = SELECT [UNIQUE] sel-expr-list
                 | SELECT [UNIQUE] *
sel-expr-list :: = sel-expr
                 | sel-expr-list, sel-expr
sel-expr :: = expr
            | var-name.*
            | table-name.*
target-list :: = host-location
               | target-list, host-location
from-list :: = table-name [var-name]
             | from-list, table-name [var-name]
field-spec-list :: = field-spec
                   | field-spec-list, field-spec
ord-spec-list :: = field-spec [direction]
                 | ord-spec-list, field-spec [direction]
direction :: = ASC | DESC
```

```
boolean  :: = boolean-term
            | boolean OR boolean-term
boolean-term :: = boolean-factor
                | boolean-term AND boolean-factor
boolean-factor :: = [NOT]  boolean-primary
boolean-primary :: = predicate
                   | ( boolean )
predicate :: = expr comparison expr
             | expr BETWEEN expr AND expr
             | expr comparison table-spec
             | < field-spec-list > = table-spec
             | < field-spec-list > [IS]  [NOT]
                                        IN table-spec
             | IF predicate THEN predicate
             | SET  ( field-spec-list ) comparison
                    table-spec
             | SET  ( field-spec-list )  comparison
                    SET  ( field-spec-list )
             | table-spec Comparison table-spec
table-spec :: = query-block
              |   ( query-expr )
              | literal
expr :: = arith-term
        | expr add-op arith-term
arith-term :: = arith-factor
              | arith-term mult-op arith-factor
arith-factor :: = [add-op] primary
```

```
primary :: = [OLD | NEW] field-spec
           |set-fn ( [UNIQUE] expr )
           |COUNT (*)
           |constant
           |( expr)
field-spec :: = field-name
              | table-name . field-name
              | var-name . field-name
comparison :: = comp-op
              | CONTAINS
              | DOES NOT CONTAIN
              |   [IS]   IN
              |   [IS]   NOT IN
comp-op :: =   = | ≠ | > | >= | < | <=
add-op :: = + | -
mult-op :: = * | /
set-fn :: = AVG | MAX | MIN | SUM | COUNT |
                                        indentifier
literal :: = ( lit-tuple-list )
           | lit-tuple
           | ( entry-list )
           | constant
lit-tuple-list :: = lit-tuple
                  | lit-tuple-list , lit-tuple
lit-tuple :: = < entry-list >
entry-list :: = entry
              | entry-list , entry
```

```
entry :: = [constant]

constant :: = quoted-string
             | number
             | host-location
             | NULL
             | USER
             | DATE
             | field-name OF CURSOR cursor-name
                     ON table-name

table-name :: = name

image-name :: = name

link-name :: = name

asrt-name :: = name

trig-name :: = name

name :: = [creator.] identifier

creator :: = identifier

user-name :: = identifier

field-name :: = identifier

var-name :: = identifier

cursor-name :: = identifier

pointer :: = identifier

save-point-name :: = identifier

host-location :: = identifier [: identifier]

integer :: = number


ddl-statement :: = create-table
                 | expand-table
```

```
                    | create-image
                    | create-link
                    | define-view
                    | define-synonym
                    | drop
                    | comment

create-table :: = CREATE TABLE table-name ( field-defn-list )

field-defn-list :: = field-defn
                    |field-defn-list , field-defn

field-defn :: = field-name ( type [, NONULL] )

type :: = CHAR (integer) [VAR]
         |INTEGER
         |SMALLINT
         |DECIMAL  (integer , [integer] )
         |FLOAT

expand-table :: = EXPAND TABLE table-name ADD COLUMN field-defn

create-image :: = CREATE [image-mod-list] IMAGE  image-name
                   ON table-name  (ord-spec-list)

image-mod-list :: = image-mod
                    |image-mod-list image-mod

image-mod :: = UNIQUE
              |CLUSTERING

create-link :: = CREATE [CLUSTERING] LINK  link-name
                  FROM table-name (field-name-list)
                  TO table-name (field-name-list)
                   [ORDER BY ord-spec-list]
```

```
define-view :: = DEFINE VIEW table-name
                 [( field-name-list )]  AS query

define-synonym :: = DEFINE-SYNONYM identifier AS table-name

drop :: = DROP system-entity name

comment :: = COMMENT ON system-entity name : quoted-string
            |COMMENT ON COLUMN table-name . field-name
                                          : quoted-string

system-entity :: = TABLE | VIEW | ASSERTION
                  |TRIGGER | IMAGE | LINK

control-statement :: = asrt-statement
                      |define-trigger
                      |grant
                      |revoke
                      |begin-trans
                      |end-trans
                      |save
                      |restore

asrt-statement :: = ASSERT asrt-name [IMMEDIATE]
                     [ON asrt-condition]  : boolean

asrt-condition :: = action-list
                   |table-name  [var-name]
```

```
action-list :: = action
              | action-list , action
action :: = INSERTION OF table-name  [var-name]
          | DELETION OF table-name  [var-name]
          | UPDATE OF table-name  [var-name]
            [( field-name-list )]
define-trigger :: = DEFINE TRIGGER trig-name
                    ON trig-condition :
                                        ( statement-list )
trig-condition :: = action
                  | READ OF table-name  [var-name]
statement-list :: = cond-statement
                  | statement-list ; cond-statement
cond-statement :: = statement
                  | IF boolean THEN statement
grant :: = GRANT  [auth]  table-name TO user-list
           [WITH GRANT OPTION]
auth :: = ALL RIGHTS ON
        | operation-list ON
        | ALL BUT operation-list ON
user-list :: = user-name
             | user-list , user-name
             | PUBLIC
operation-list :: = operation
                  | operation-list , operation
operation :: = READ
             | INSERT
```

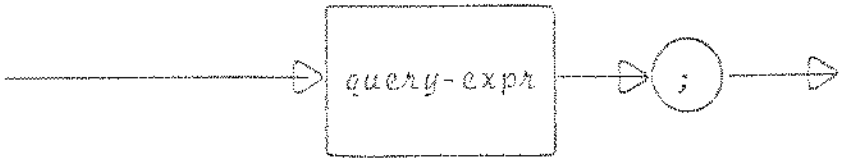
```
                | DELETE
                | UPDATE  [(field-name-list)]
                | EXPAND
                | IMAGE
                | LINK
                | CONTROL
                | RUN
revoke :: = REVOKE [operation-list ON] table-name
                    FROM user-list
begin-trans :: = BEGIN TRANSACTION
end-trans :: = END TRANSACTION
save :: - SAVE save-point-name
restore :: = RESTORE  [save-point-name]

cursor-statement :: = let
                    | open
                    | fetch
                    | close
                    | describe
                    | execute
let :: = LET cursor-name BE query
       | LET cursor-name BE execute
open :: = OPEN cursor-name
fetch :: = FETCH cursor-name  [INTO pointer]
close :: = CLOSE cursor-name
describe :: = DESCRIBE cursor-name INTO pointer
execute :: = EXECUTE host-location
```

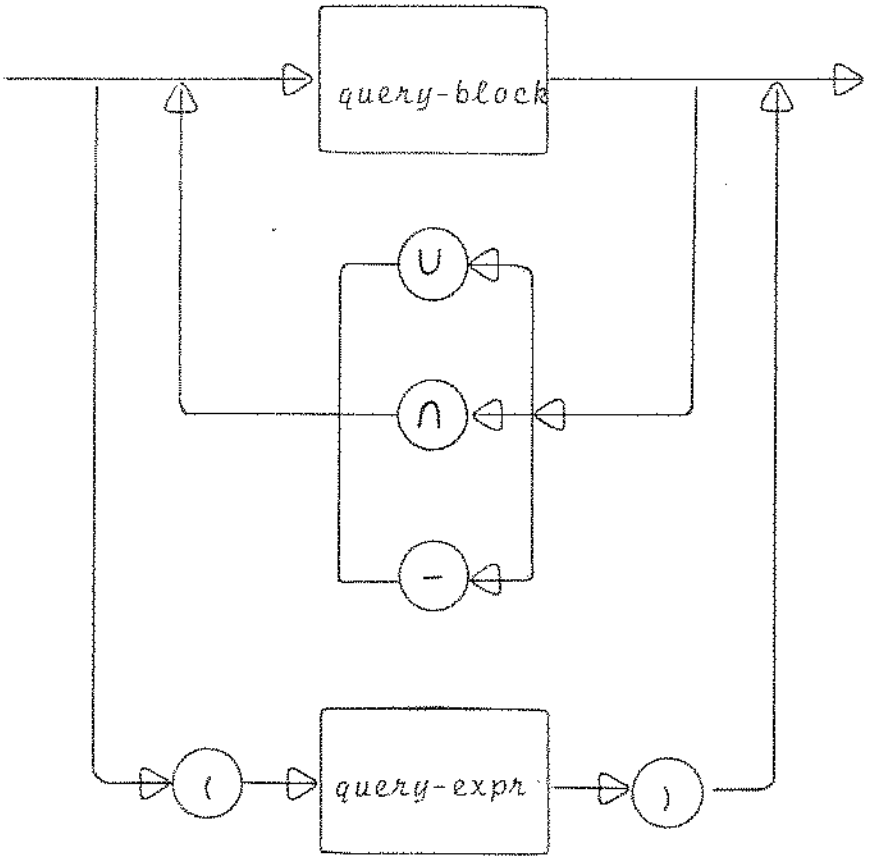
APÊNDICE B

MAPA SINTÁTICO DO SUBCONJUNTO DE SEQUEL 2 IMPLEMENTADO NO BIBLOS

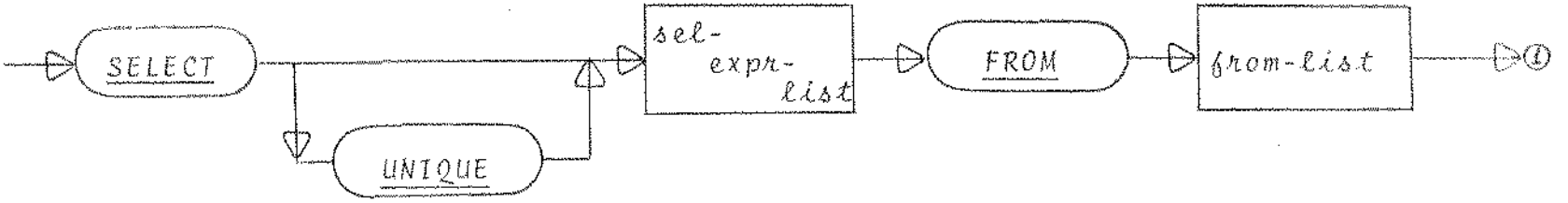
QUERY

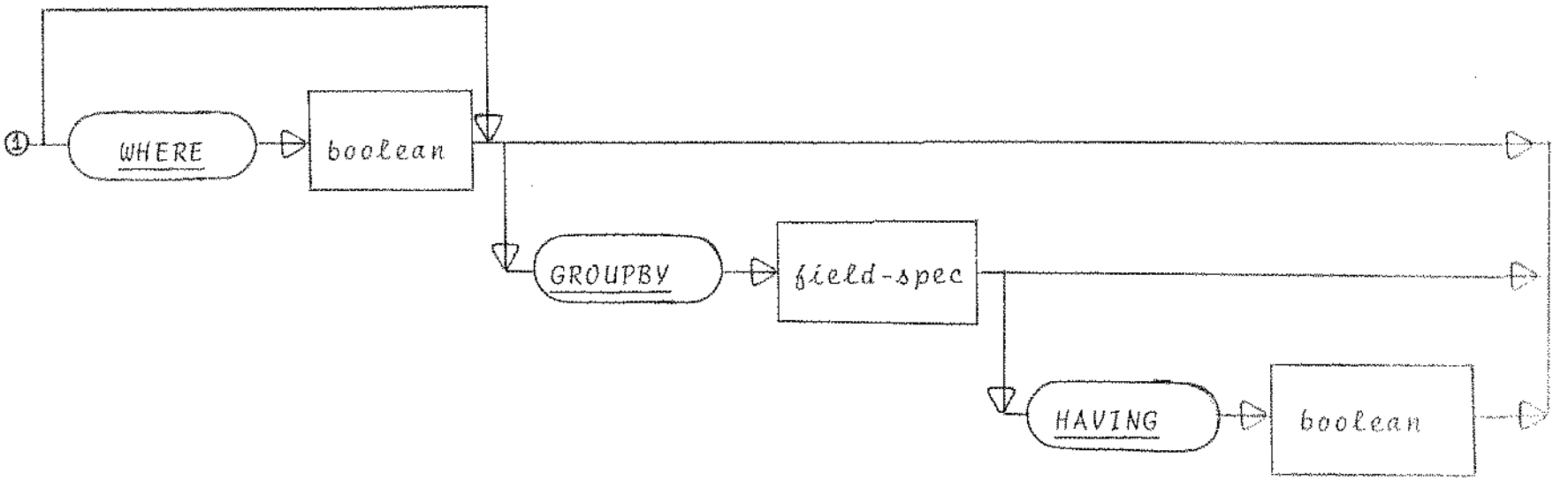


QUERY-EXPR

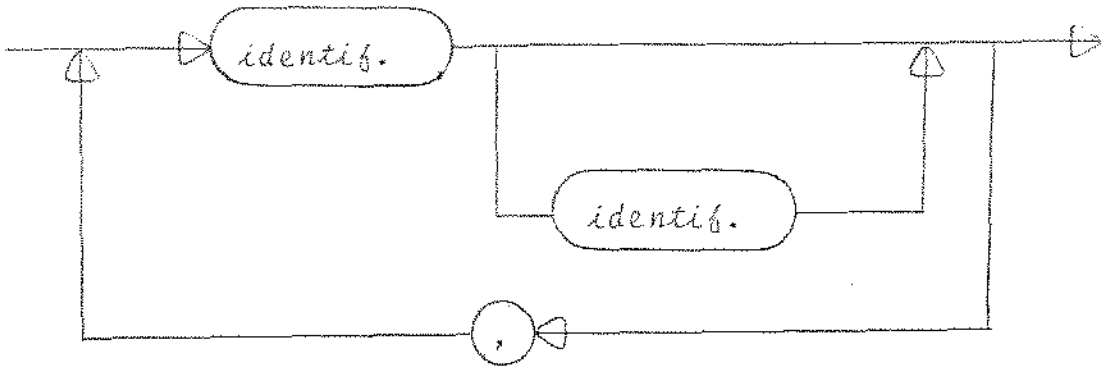


QUERY-BLOCK

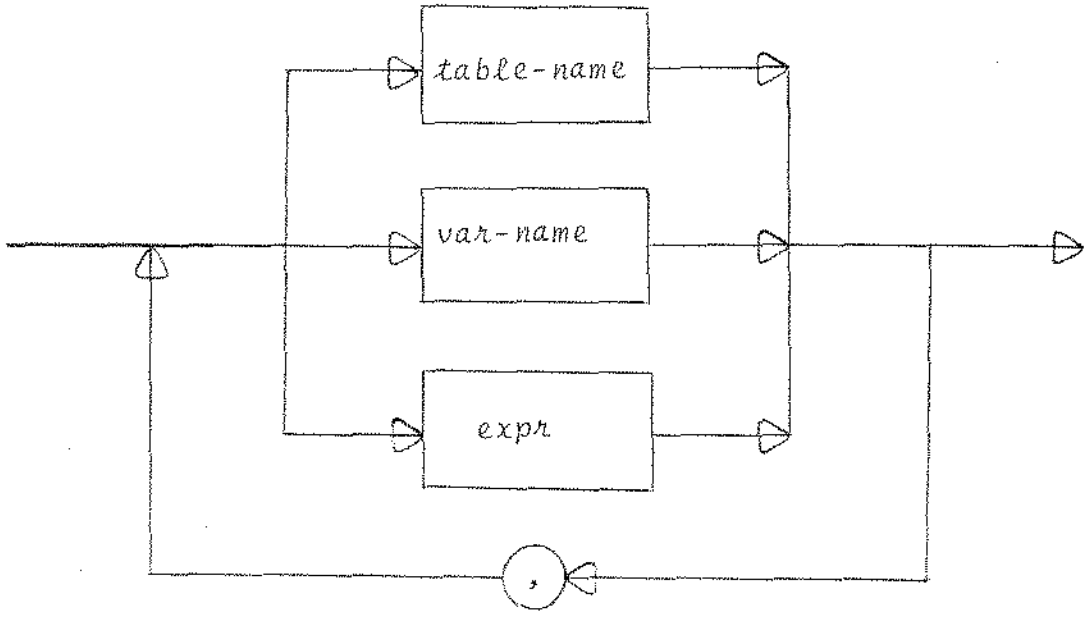

SELECT
UNIQUE
sel-
expr-
list
FROM
from-list
①


①
WHERE
boolean
GROUPBY
field-spec
HAVING
boolean

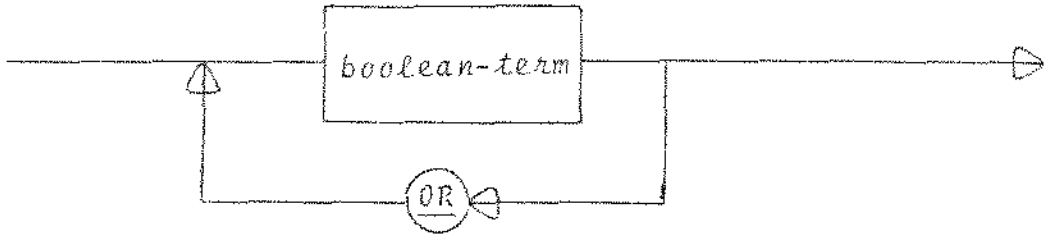
FROM-LIST



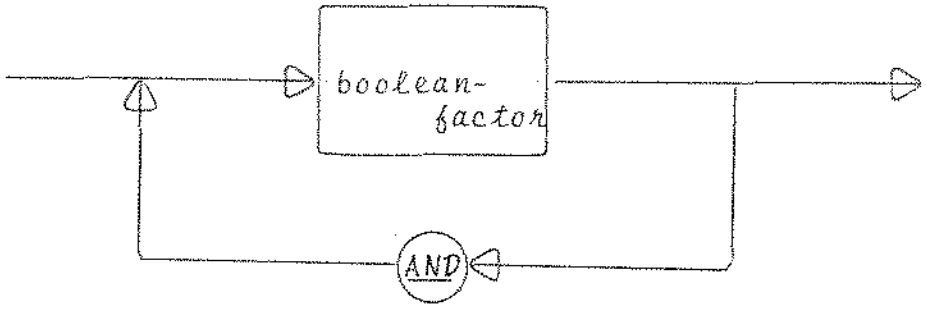
SEL-EXPR-LIST



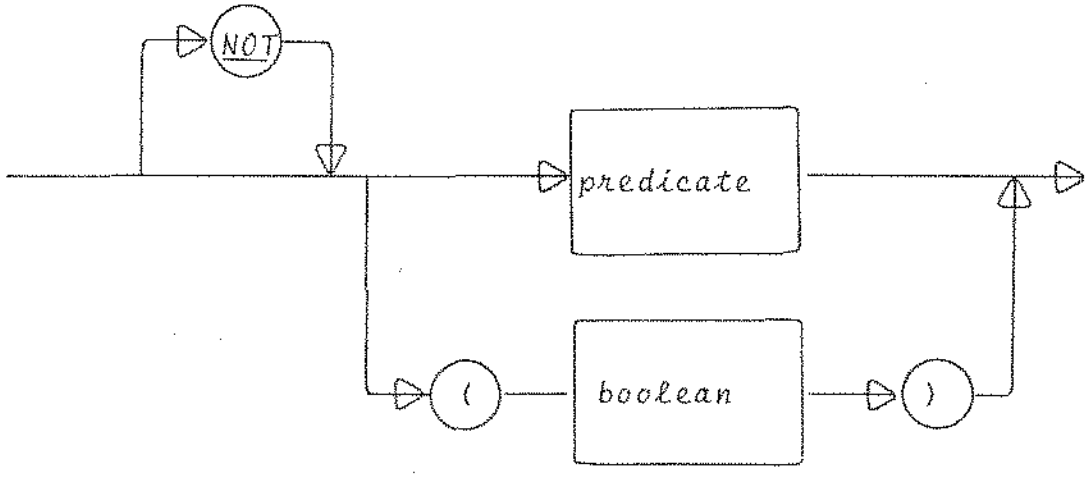
BOOLEAN



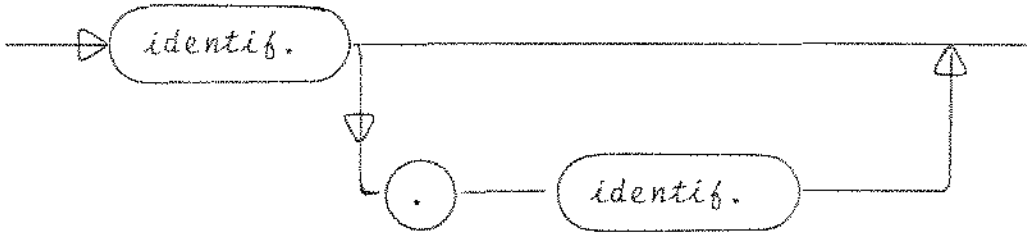
BOOLEAN-TERM



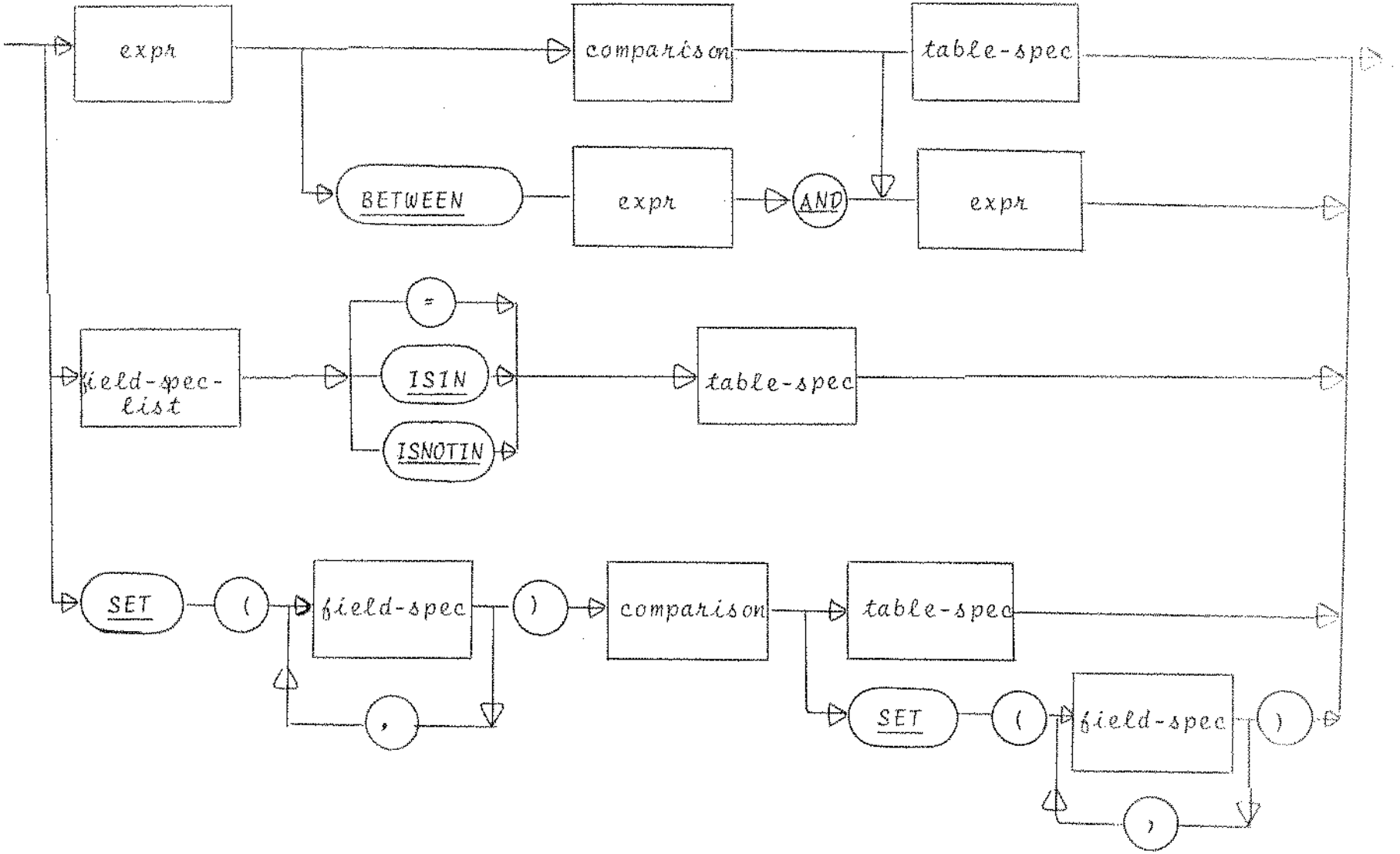
BOOLEAN-FACTOR



FIELD-SPEC



PREDICATE


expr
comparison
table-spec
BETWEEN
expr
AND
expr
field-spec-list
=
ISIN
ISNOTIN
table-spec
SET
(
field-spec
)
comparison
table-spec
,
SET
(
field-spec
)
)

COMPARISON

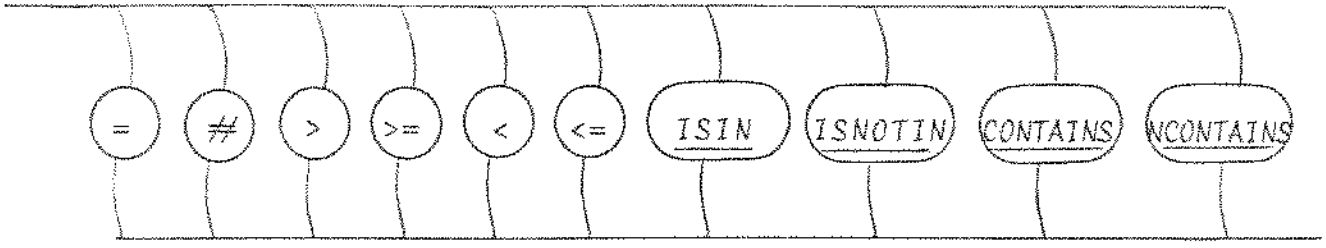
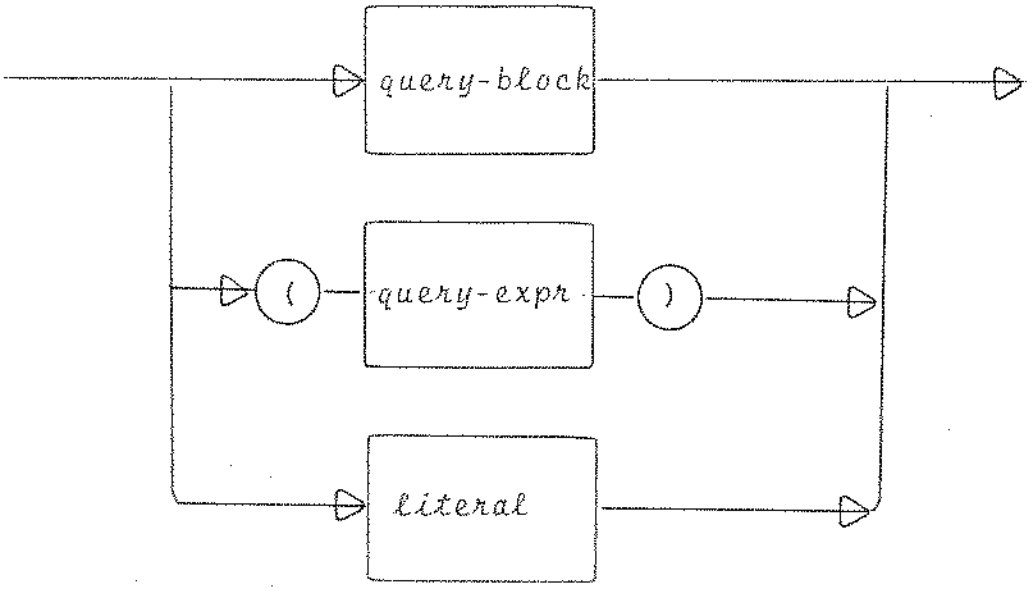
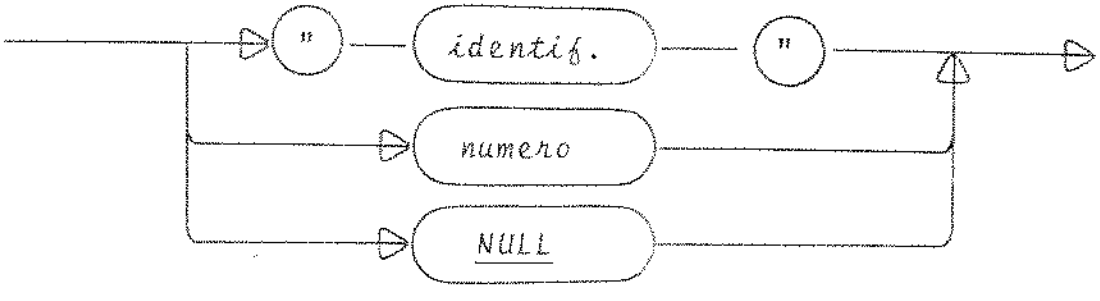


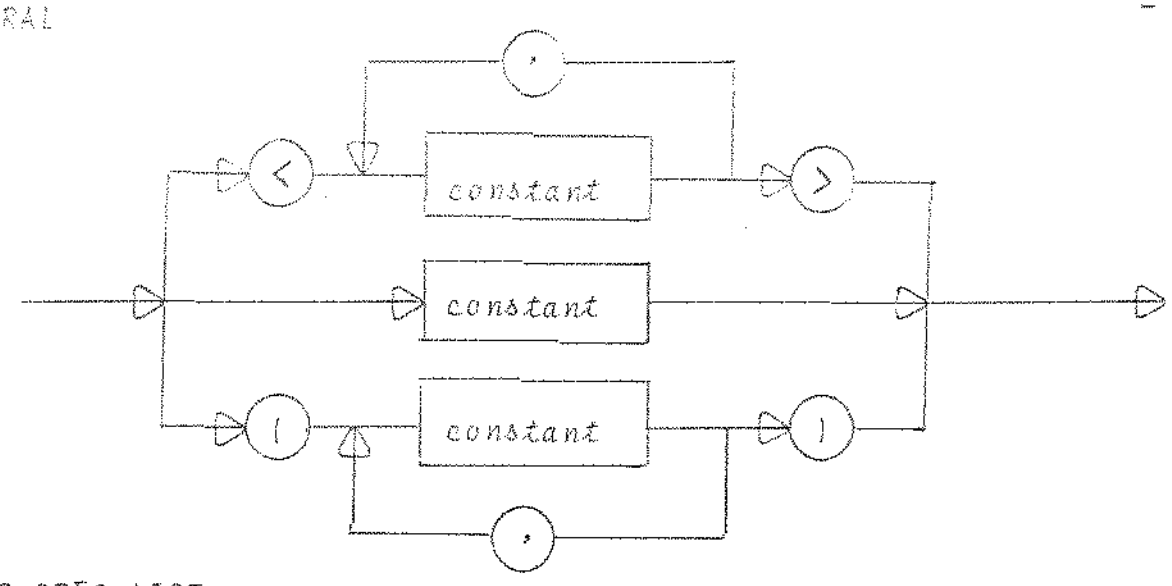
TABLE-SPEC



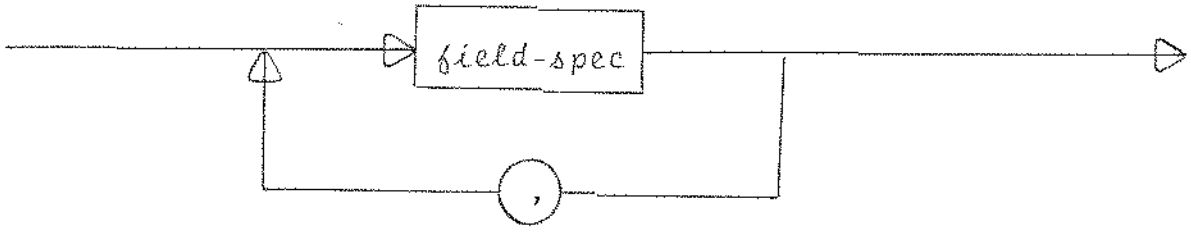
CONSTANT



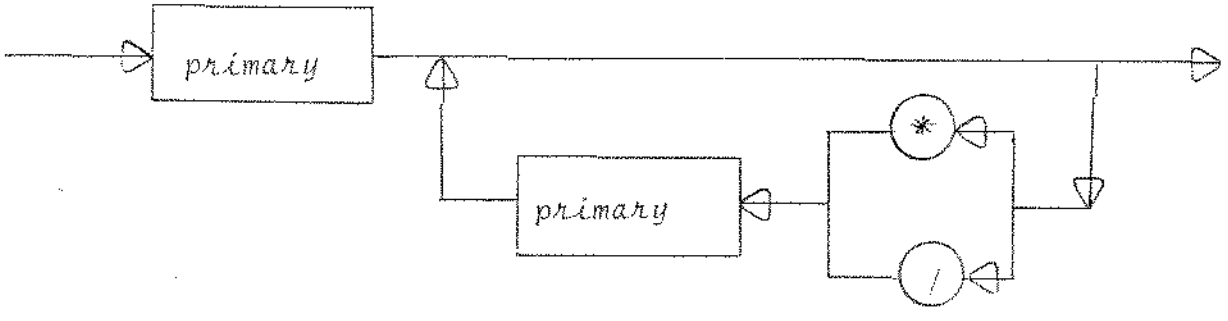
LITERAL



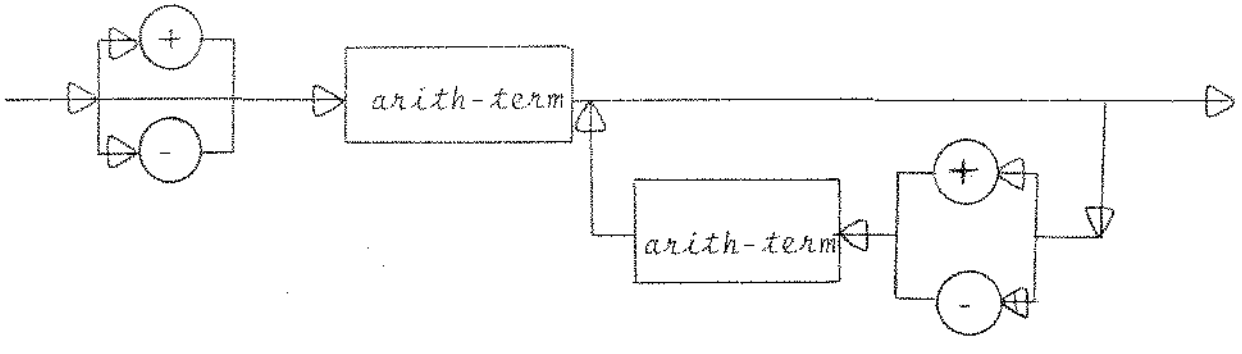
FIELD-SPEC-LIST



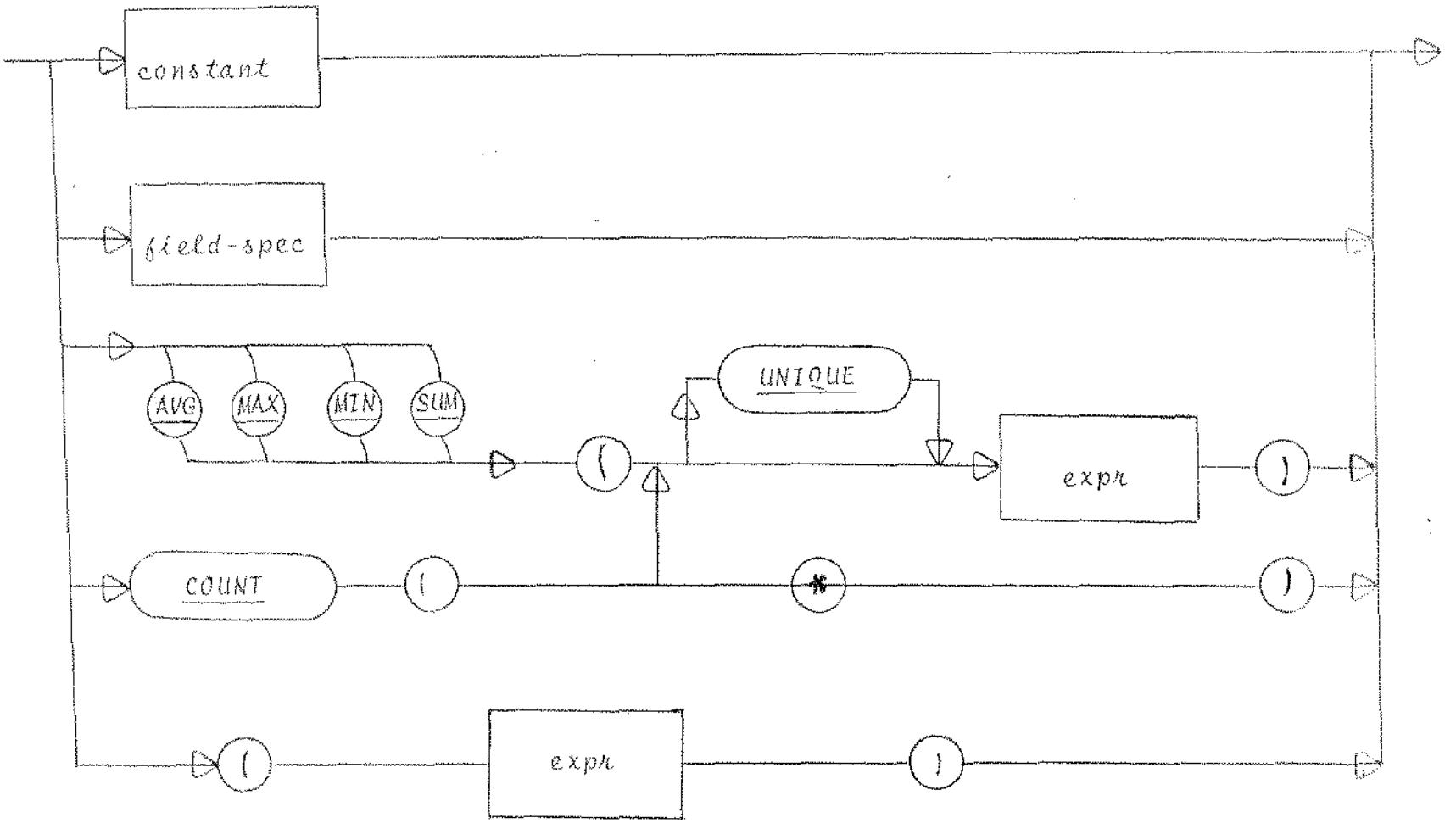
ARITH-TERM



EXPR



PRIMARY



APÊNDICE C

USO DA LINGUAGEM SEQUEL2 UTILIZANDO UM BANCO DE DADOS DE DEMONSTRAÇÃO.

Conteúdo das Relações citadas na Fig. 3.12 utilizadas como demonstração da linguagem SEQUEL2

EMP

NAME	SALARY	MGR	DEPT	COMM
Jones	8000	Smith	Household	4000
Anderson	6000	Murphy	Toy	3000
Morgan	10000	Long	Cosmetics	5000
Lewis	12000	Long	Stationery	6000
Nelson	6000	Murphy	Toy	0
Hoffman	16000	Morgan	Cosmetics	0
Long	7000	Morgan	Cosmetics	3500
Murphy	8000	Smith	Household	4000
Smith	12000	Hoffman	Stationery	6000
Henry	9000	Smith	Toy	4500

SALES

DEPT	ITEM
Stationery	Dish
Household	Pen
Stationery	Pencil
Cosmetics	Lipstick
Toy	Pen
Toy	Pencil
Toy	Ink
Cosmetics	Perfume
Stationery	Ink
Household	Disk
Stationery	Pen
Hardware	Ink

TYPE

ITEM	COLOR	SIZE
Disk	White	M
Lipstick	Red	L
Perfume	White	L
Pen	Green	S
Pencil	Blue	M
Ink	Green	L
Ink	Blue	S
Pencil	Red	L
Pencil	Blue	L

SUPPLY

ITEM	SUPPLIER
Pen	Pencraft
Pencil	Flic
Ink	Pencraft
Perfume	Beautex
Ink	Flic
Dish	Chemco
Lipstick	Beautex
Disk	Flic
Pen	Beautex
Pencil	Pencraft

```
.RUN SEQUEL

*

* SELECT COLOR
    FROM TYPE
  WHERE ITEM ISIN <"DISH","PEN">;

> CONSULTA <C> MANUTENCAO (M) REMOCAO (R) :

> DISCO (D) DISPLAY <T> :

** RESPOSTA **

 WHITE  /
 GREEN  /

* SELECT UNIQUE COLOR
    FROM TYPE;

> CONSULTA <C> MANUTENCAO (M) REMOCAO (R) : C

> DISCO (D) DISPLAY <T> : T

** RESPOSTA **

 WHITE  /
 RED    /
 GREEN  /
 BLUE   /

* SELECT TYPE
    FROM TYPE;

> CONSULTA <C> MANUTENCAO (M) REMOCAO (R) :

> DISCO (D) DISPLAY <T> :

** RESPOSTA **

 DISH      WHITE     M       /
 LIPSTICK  RED       L       /
 PERFUME   WHITE     L       /
 PEN       GREEN     S       /
 PENCIL    BLUE      M       /
 INK       GREEN     L       /
 INK       BLUE      S       /
 PENCIL    RED       L       /
 PENCIL    BLUE      L       /
```

```
* SELECT ITEM
     FROM TYPE
  WHERE COLOR = "BLUE";


> CONSULTA <C> MANUTENCAO (M) REMOCAO (R) :

> DISCO (D) DISPLAY <T> :


** RESPOSTA **

 PENCIL /
 INK    /
 PENCIL /


* SELECT UNIQUE ITEM
     FROM TYPE
  WHERE COLOR = "BLUE";


> CONSULTA <C> MANUTENCAO (M) REMOCAO (R) :

> DISCO (D) DISPLAY <T> :


** RESPOSTA **

 PENCIL /
 INK    /


* SELECT NAME
     FROM EMP
  WHERE SALARY ISIN <8000,12000,9000>;


> CONSULTA <C> MANUTENCAO (M) REMOCAO (R) :

> DISCO (D) DISPLAY <T> :


** RESPOSTA **

 JONES  /
 LEWIS  /
 MURPHY /
 SMITH  /
 HENRY  /


* SELECT NAME
     FROM EMP
  WHERE DEPT ISIN
     SELECT DEPT
        FROM SALES
     WHERE ITEM = "DISH";
```

```
> CONSULTA <C> MANUTENCAO (M) REMOCAO (R) :

> DISCO (D) DISPLAY <T> :


** RESPOSTA **

 JONES  /
 LEWIS  /
 MURPHY /
 SMITH  /


* SELECT NAME
    FROM EMP
  WHERE DEPT = "TOY" AND SALARY > 10000;

NENHUM VALOR ENCONTRADO QUE SATISFACA A CONSULTA


* SELECT AVG(SALARY), NAME
    FROM EMP
  WHERE DEPT = "HOUSEHOLD";


> CONSULTA <C> MANUTENCAO (M) REMOCAO (R) :

> DISCO (D) DISPLAY <T> :


** RESPOSTA **

 8000     JONES  /
 8000     MURPHY /


* SELECT AVG(SALARY)
    FROM EMP
  WHERE DEPT = "HOUSEHOLD";


> CONSULTA <C> MANUTENCAO (M) REMOCAO (R) :

> DISCO (D) DISPLAY <T> :


** RESPOSTA **

 8000   /


* SELECT AVG(SALARY)/52
    FROM EMP
  WHERE MGR = "MORGAN";


> CONSULTA <C> MANUTENCAO (M) REMOCAO (R) :
```

```
> DISCO (D) DISPLAY <T> :

** RESPOSTA **

 221     /


* SELECT COUNT(UNIQUE DEPT)
     FROM EMP
   WHERE MGR = "MURPHY";


> CONSULTA <C> MANUTENCAO (M) REMOCAO (R) ;

> DISCO (D) DISPLAY <T> :


** RESPOSTA **

 1       /


* SELECT DEPT
     FROM EMP
   GROUPBY DEPT;

> DISCO (D) DISPLAY <T> :


** RESPOSTA **

 COSMETICS       /
 HOUSEHOLD       /
 STATIONERY      /
 TOY             /


* SELECT DEPT
     FROM EMP
   GROUPBY DEPT
   HAVING AVG(SALARY) > 10000;

> DISCO (D) DISPLAY <T> :


** RESPOSTA **

 COSMETICS       /
 STATIONERY      /


* SELECT DEPT
     FROM EMP
   WHERE MGR = "SMITH"
   GROUPBY DEPT
   HAVING COUNT(*) > 2;


NENHUM VALOR ENCONTRADO QUE SATISFACA A CONSULTA
```

```
* SELECT DEPT
    FROM EMP
  SUBTRACT
  SELECT DEPT
    FROM SALES
  WHERE ITEM ISNOTIN
    SELECT ITEM
      FROM SUPPLY
    WHERE SUPPLIER = "PENCRAFT";

> DISCO (D) DISPLAY <T> :


** RESPOSTA **

 IOY     /


* SELECT NAME, SALES.DEPT
    FROM EMP, SALES
  WHERE SALES.ITEM = "DISH"" AND EMP.DEPT = SALES.DEPT;

> DISCO (D) DISPLAY <T> :


** RESPOSTA **

 JONES    HOUSEHOLD       /
 LEWIS    STATIONERY      /
 MURPHY   HOUSEHOLD       /
 SMITH    STATIONERY      /


* SELECT X.NAME, Y.NAME
    FROM EMP X, EMP Y
  WHERE X.MGR = Y.NAME AND X.SALARY > Y.SALARY;

> DISCO (D) DISPLAY <T> :


** RESPOSTA **

 MORGAN          LONG    /
 LEWIS           LONG    /
 HOFFMAN         MORGAN  /


* SELECT NAME
    FROM EMP
  WHERE <DEPT,SALARY> ISIN
    SELECT DEPT, SALARY
      FROM EMP
    WHERE SALARY = 6000;


> CONSULTA <C> MANUTENCAO (M) REMOCAO (R) :

> DISCO (D) DISPLAY <T> :
```

```
** RESPOSTA **

 ANDERSON        /
 NELSON          /


* SELECT TYPE.ITEM
     FROM TYPE, SALES
  WHERE TYPE.ITEM = SALES.ITEM AND COLOR = "GREEN" AND DEPT = "TOY";

> DISCO (D) DISPLAY <T> :


** RESPOSTA **

 PEN     /
 INK     /


* SELECT NAME, SALARY
     FROM EMP
  WHERE DEPT ISIN
     (SELECT DEPT FROM SALES WHERE ITEM = "PEN")
  AND SALARY = 8000;


> CONSULTA <C> MANUTENCAO (M) REMOCAO (R) :

> DISCO (D) DISPLAY <T> :


** RESPOSTA **

 JONES     8000  /
 MURPHY    8000  /


* SELECT SALES.DEPT, SUPPLIER
     FROM SALES X, SUPPLY Y
  WHERE X.ITEM = Y.ITEM;

> DISCO (D) DISPLAY <T> :


** RESPOSTA **

 STATIONERY        CHEMCO    /
 STATIONERY        FLIC      /
 HOUSEHOLD         PENCRAFT  /
 HOUSEHOLD         BEAUTEX   /
 STATIONERY        FLIC      /
 STATIONERY        PENCRAFT  /
 COSMETICS         BEAUTEX   /
 TOY               PENCRAFT  /
 TOY               BEAUTEX   /
 TOY               FLIC      /
 TOY               PENCRAFT  /
 TOY               PENCRAFT  /
```

```
TOY                 FLIC      /
COSMETICS           BEAUTEX   /
STATIONERY          PENCRAFT  /
STATIONARY          FLIC      /
HOUSEHOLD           CHROMCO   /
HOUSEHOLD           FLIC      /
STATIONERY          PENCRAFT  /
STATIONERY          BEAUTEX   /
HARDWARE            PENCRAFT  /
HARDWARE            FLIC      /


* SELECT NAME
     FROM EMP
  WHERE SALARY BETWEEN 10000 AND 15000 AND SALARY # 13000;


> CONSULTA <C> MANUTENCAO (M) RENOCAO (R) :

> DISCO (D) DISPLAY <T> :


** RESPOSTA **

 LEWIS  /
 SMITH  /


* SELECT DEPT
     FROM SALES
  GROUPBY DEPT
  HAVING SET(ITEM) CONTAINS
     SELECT ITEM FROM TYPE WHERE COLOR = "GREEN";

> DISCO (D) DISPLAY <T> :


** RESPOSTA **

 STATIONERY     /
 TOY            /


* SELECT DEPT
     FROM SALES
  GROUPBY DEPT
  HAVING SET(ITEM) ISIN
     SELECT ITEM FROM TYPE WHERE COLOR = "GREEN";

> DISCO (D) DISPLAY <T> :


** RESPOSTA **

 HARDWARE       /


* SELECT SIZE FROM TYPE WHERE ITEM = "DISH" AND COLOR = "WHITE";
```

```
> CONSULTA <C> MANUTENCAO (M) REMOCAO (R) : M

TUPLA = 000001*01LISH*02WHITE*03M #

NOVO VALOR PARA SIZE       (03) : L

* SELECT ITEM FROM SUPPLY WHERE SUPPLIER = "FII" AND IIEM = "PENCIL";

> CONSULTA <C> MANUTENCAO (M) REMOCAO (R) : M

TUPLA = 000002*01PENCIL*02FLIC #

NOVO VALOR PARA ITEM       (01) : PEN

* SELECT ITEM FROM SALES WHERE DEPT = "TOY" AND ITEM = "INK";

> CONSULTA <C> MANUTENCAO (M) REMOCAO (R) : M

TUPLA = 000007*01TOY*02INK #

NOVO VALOR PARA ITEM       (02) : BOX

* SELECT COMM FROM EMP WHERE NAME = "ANDERSON";

> CONSULTA <C> MANUTENCAO (M) REMOCAO (R) : M

TUPLA = 000002*01ANDERSON*026000*03MURPHY*04TOY*053000 #

NOVO VALOR PARA COMM       (05) : 4500

* SELECT SUPPLIER FROM SUPPLY WHERE SUPPLIER = "REAUTEX";

> CONSULTA <C> MANUTENCAO (M) REMOCAO (R) : R

* *
  FIM DAS CONSULTAS

.

.

.

. TYPE DELETE
        4SUP
        7SUP
        9SUP

.TYPE INSERT
       1TYP01 DISH
```

```
        1TYP02 WHITE
        1TYP03 L
&
        2SUP01 PEN
        2SUP02 FLTC
&
        7SAL01 TOY
        7SAL02 BOX
&
        2EMP01 ANDERSON
        2EMP02 6000
        2EMP03 MURPHY
        2EMP04 TOY
        2EMP05 4500
&
```

```
.RU BLBINS

                * *  SISTEMA BIBLOS  * *

          * INCLUSAO DE REGISTROS EM TABELAS *

> IDENTIFICACAO :

> TABELA (SSS) : TYP

OK - FIM DA INSERCAO

.RU BLBINS

                * *  SISTEMA BIBLOS  * *

          * INCLUSAO DE REGISTROS EM TABELAS *

> IDENTIFICACAO :

> TABELA (SSS) : SUP

OK - FIM DA INSERCAO

.RU BLBINS

                * *  SISTEMA BIBLOS  * *

          * INCLUSAO DE REGISTROS EM TABELAS *

> IDENTIFICACAO :

> TABELA (SSS) : SAL
```

```
OK - FIM DA INSERCAO

-RU BURINS

                * *  SISTEMA BIBLUS  * *

          * INCLUSAO DE REGISTROS EM TABELAS *

> IDENTIFICACAO :

> TABELA (SSS) : EMP

OK - FIM DA INSERCAO

-RU BURDEL

                * *  SISTEMA BIBLUS  * *

          * DESATIVACAO DE TUPLAS EM TABELAS *

> IDENTIFICACAO :

> TABELA (SSS) : SUP

OK - FIM DA REMOCAO

-RU SEQUEL

*

* SELECT SIZE
    FROM TYPE
  WHERE ITEM = "DISH" AND COLOR = "WHITE";

> CONSULTA <C> MANUTENCAO (M) REMOCAO (P) :

> DISCO (D) DISPLAY <T> : T

** RESPOSTA **

 L       /

* SELECT ITEM
    FROM SUPPLY
  WHERE SUPPLIER = "FLIC";
```

```
> CONSULTA <C> MANUTENCAO (M) REMOCAO (R) :

> DISCO (D) DISPLAY <T> : T

** RESPOSTA **

 PEN     /
 INK     /
 DISH    /

* SELECT ITEM FROM SALES WHERE DEPT = "TOY";

> CONSULTA <C> MANUTENCAO (M) REMOCAO (R) :

> DISCO (D) DISPLAY <T> :

** RESPOSTA **

 PEN     /
 PENCIL  /
 BOX     /

* SELECT NAME, SALARY, COMM FROM EMP WHERE NAME = "ANDERSON";

> CONSULTA <C> MANUTENCAO (M) REMOCAO (R) :

> DISCO (D) DISPLAY <T> :

** RESPOSTA **

 ANDERSON          6000    4500   /

* SELECT SUPPLIER FROM SUPPLY WHERE SUPPLIER = "BEAUTEX";

NENHUM VALOR ENCONTRADO QUE SATISFACA A CONSULTA

* @

 FIM DAS CONSULTAS

.
```

APÊNDICE D

DEFINIÇÕES DE ALGUMAS OPERAÇÕES UTILIZADAS NA MANIPULAÇÃO DE RELAÇÕES.

D1 - PRODUTO CARTESIANO

O produto cartesiano RxS de uma relação R (grau m) com a relação S (grau n) é uma relação de grau m+n. Este produto é definido por

$$R \otimes S = \left\{ (r \frown s) : r \in R \wedge s \in S \right\}$$

(o símbolo $\frown$ é usado para designar a concatenação das tuplas r e s)

D2 - PROJEÇÃO

Suponha que r seja uma tupla da relação R de grau m. Para j= 1,2, ...,m a notação $r\left[j\right]$ designa o j-ésimo componente de r. Para outros valores de j, $r\left[j\right]$ não é definido. Estende - se a notação para uma lista $A=(j_1, j_2, \ldots, j_r)$ de inteiros (não necessariamente distintos) do conjunto 1,2, ...,m como se segue:

$$r\left[A\right] = (r\left[j_1\right], r\left[j_2\right], \ldots, r\left[j_r\right]).$$

Quando a lista A é vazia, $r\left[A\right] = r$. Seja R uma relação de grau m, e A uma lista de inteiros (não necessariamente distintos) do conjunto 1,2, ..., m . Então a projeção de R em A é definida como

$$R\left[A\right] = \left\{ r\left[A\right] : r \in R \right\}$$

Quando A é uma permutação da lista (1,2, ..., m), $R\left[A\right]$ é uma / relação cujos domínios são os mesmos de R exceto pela mudança na ordem em que aparecem. Na figura D2 aparecem alguns exemplos de / projeções.

R (D_1	D_2	D_3)
a	2	f
b	1	g
c	3	f
d	3	g
e	2	f

$R[3]$ (D_1)
f
g

$R[3,2,2]$ (D_1	D_2	D_3)
f	2	2
g	1	1
f	3	3
g	3	3

Fig. D2 - Uma relação R e duas de suas muitas projeções.

D3 - JOIN

Suponha que o símbolo Θ represente uma das relações $=$, $\neq$, $<$, $\leq$, $>$ e $\geq$. A operação Θ-join da relação R no domínio A com a relação S no domínio B é definido por

$$R[A \Theta B] S = \{(r \frown s): r \in R \wedge s \in S \wedge (r[A] \Theta s[B])\}$$

desde que todo elemento de $R[A]$ seja Θ-comparável com todo elemento de $S[B]$. Note que x é Θ-comparável com y se $x \Theta y$ é verdadeiro ou falso (não indefinido). Na figura D3 aparecem al- / guns exemplos de joins.

R (A	B	C)
a	1	1
a	2	1
b	1	2
c	2	5
c	3	3

S (D	E)
2	u
3	v
4	u

R [C = D] S

A	B	C	D	E
b	1	2	2	u
c	3	3	3	v

R [C > D] S

A	B	C	D	E
c	3	3	2	u
c	2	5	2	u
c	2	5	3	v
c	2	5	4	u

Fig. D3 - Relações R,S e dois exemplos de joins.

A operação de join mais comum é o join quando Θ é igual a '=', chamado equi-join.

D4 - RESTRIÇÃO

Suponha que R é uma relação e A, B são listas que identificam os domínios de R. Seja Θ uma das relações =, $\neq$, <, $\leq$, > e $\geq$. A operação Θ-restrição em R nos dominios A,B é definida como

$$R\,[A\ \Theta\ B] = \{r: r \in R \wedge (r[A]\ \Theta\ r[B])\}$$

desde que todo elemento de R[A] seja Θ-comparável com todo elemento de R[B]. Na figura D4 aparecem alguns exemplos de restrições.

R

A	B	C
p	2	1
q	2	3
q	5	4
r	3	3

R [B = C]

A	B	C
r	3	3

R [B > C]

A	B	C
p	2	1
q	5	4

Fig. D4 - Relação R e duas de suas restrições.