


UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL  
PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

ANÁLISE COMPARATIVA DAS FORMAS E MÉTODOS DE  
EMBUTIMENTO DE LINGUAGENS PARA MANIPULAÇÃO DE  
BANCO DE DADOS

por

HUBERT AHLERT

Dissertação submetida como requisito parcial  
para a obtenção do grau de Mestre em  
Ciência da Computação

  
Profª Lia Goldstein Golendziner  
Orientadora

Porto Alegre, março de 1984

Ahlert, Hubert

Análise comparativa das formas e métodos de embutimento de linguagens para manipulação de banco de dados. Porto Alegre, PGCC da UFRGS, 1984.  
298f.

Diss. (mestr. ci. comp.) UFRGS-PGCC, Porto Alegre, BR-RS, 1984.

Dissertação:

- . Linguagens de banco de dados
- . Embutimento de linguagens

## AGRADECIMENTOS

À Professora Lia Goldstein Golendziner, orientadora deste trabalho, pelas valiosas críticas e sugestões apresentadas no decorrer do trabalho e pela atenção com que acolheu esta dissertação.

Ao Professor Carlos Alberto Heuser, pela motivação inicial, além das sugestões fornecidas e informações prestadas na fase de definição do assunto da presente dissertação.

Aos Professores Clesio Saraiva dos Santos, José Mauro Volkmer de Castilho e Paulo Alberto de Azeredo pela colaboração e contribuições concedidas quando da apresentação do seminário sobre o andamento do trabalho.

À senhora Carmem Souza Souza, secretária do curso, pelo suporte para a obtenção do grau de mestre.

Às bibliotecárias Zita Catarina P. de Oliveira e Margarida C.S. Buchmann pelo apoio concedido na tarefa de levantamento e revisão bibliográfica.

Às amigas Neia Sheila Moraes de Moraes e Helenara H. da Silva Fão pela dedicação com que confeccionaram as ilustrações, e também à senhora Maria Helena A. Cardozo pelos serviços de datilografia prestados.

Ao Centro de Processamento de Dados da UFRGS pelo apoio e condições oferecidas durante o desenvolvimento da dissertação.

A todos que, de alguma forma, colaboraram para a elaboração deste trabalho.

Ao meu pai, minha esposa e demais familiares pelo apoio e compreensão recebidos e, principalmente, à minha mãe que, embora não presente, SEMPRE procurou incentivar meu estudo e aperfeiçoamento profissional.

## SUMÁRIO

RESUMO .....	15
ABSTRACT .....	16
1. INTRODUÇÃO .....	17
2. INTERFACE DE MANIPULAÇÃO DE DADOS NO CONTEXTO DA ARQUITETURA DE UM S.G.B.D. ....	21
3. EMBUTIMENTO DE LINGUAGENS DE MANIPULAÇÃO DE UM S.G.B.D. ....	25
3.1 Considerações preliminares .....	25
3.2 Preocupações no projeto do embutimento .....	26
3.2.1 Interface entre o S.G.B.D. e o ambien- te da linguagem .....	26
3.2.2 Homogeneidade .....	30
3.2.3 Tipos de dados e conversões .....	31
3.2.4 Estruturas de controle de fluxo .....	33
3.2.5 Tratamento de exceções .....	34
3.2.6 Formas de implementar o embutimento ..	35
4. ESTUDO DE ALTERNATIVAS PARA INTERFACE COM B.D. BA- SEADA EM ASPECTOS HUMANOS .....	39
4.1 Considerações preliminares .....	39
4.2 Considerações sobre o projeto da interface ..	40
4.3 Questionamentos sobre aspectos humanos no pro- jeto de uma linguagem de Banco de Dados .....	41
4.4 Ferramentas utilizadas para mensurar facili- dades de linguagens .....	47
4.5 Breves comentários relativos a experimentos sobre linguagens .....	50
5. ANÁLISE DE LINGUAGENS PARA MANIPULAÇÃO DE BANCOS DE DADOS (PRINCIPAIS S.G.B.D.) .....	52
5.1 Considerações preliminares .....	52
5.2 O Sistema TOTAL .....	57
5.2.1 Caracterização do sistema .....	57
5.2.1.1 Introdução .....	57
5.2.1.2 Estruturas de dados .....	57
5.2.1.3 Linguagens hospedeiras .....	58
5.2.1.4 Estrutura funcional .....	58

5.2.2	Interface de definição dos dados no sistema	59
5.2.2.1	Definição da base de dados ..	59
5.2.2.2	Definição de esquemas externos	63
5.2.3	Aspectos funcionais do embutimento ...	64
5.2.3.1	Comunicação usuário com o Banco de Dados .....	64
5.2.3.2	Conversões .....	72
5.2.3.3	Estruturas de controle de fluxo	73
5.2.3.4	Considerações sobre proteção dos dados .....	73
5.2.3.5	Considerações sobre homogeneidade	75
5.2.4	Aspectos de implementação do embutimento ....	75
5.2.4.1	Descrição da forma de embutimento	75
5.2.4.2	Considerações sobre a portabilidade da linguagem .....	76
5.3	O Sistema ADABAS .....	77
5.3.1	Caracterização do sistema .....	77
5.3.1.1	Introdução .....	77
5.3.1.2	Estruturas de dados .....	77
5.3.1.3	Linguagens hospedeiras .....	78
5.3.1.4	Estrutura funcional .....	79
5.3.2	Interface de definição dos dados no sistema	81
5.3.2.1	Definição da base de dados ..	81
5.3.2.2	Definição de esquemas externos ...	84
5.3.3	Aspectos funcionais do embutimento ...	85
5.3.3.1	Comunicação usuário com o Banco de Dados .....	85
5.3.3.2	Conversões .....	94
5.3.3.3	Estruturas de controle de fluxo	94
5.3.3.4	Considerações sobre proteção dos dados .....	95
5.3.3.5	Considerações sobre homogeneidade	96
5.3.4	Aspectos de implementação do sistema .	97
5.3.4.1	Descrição da forma do embutimento	97
5.3.4.2	Considerações sobre a portabilidade da linguagem .....	98

5.4	O Sistema IMS .....	99
5.4.1	Caracterização do sistema .....	99
5.4.1.1	Introdução .....	99
5.4.1.2	Estruturas de dados .....	99
5.4.1.3	Linguagens hospedeiras .....	101
5.4.1.4	Estrutura funcional .....	101
5.4.2	Interface de definição dos dados no sistema .....	102
5.4.2.1	Definição da base de dados ..	102
5.4.2.2	Definição de esquemas externos ...	107
5.4.3	Aspectos funcionais do embutimento ...	108
5.4.3.1	Comunicação usuário com o Banco de Dados .....	108
5.4.3.2	Conversões .....	118
5.4.3.3	Estruturas de controle de fluxo ..	119
5.4.3.4	Considerações sobre proteção dos dados .....	119
5.4.3.5	Considerações sobre homogeneidade .....	120
5.4.4	Aspectos de implementação do embutimento ....	121
5.4.4.1	Descrição da forma de embutimento .....	121
5.4.4.2	Considerações sobre a portabilidade da linguagem .....	123
5.5	O Sistema IDMS .....	124
5.5.1	Caracterização do sistema .....	124
5.5.1.1	Introdução .....	124
5.5.1.2	Estruturas de dados .....	124
5.5.1.3	Linguagens hospedeiras .....	125
5.5.1.4	Estrutura funcional .....	125
5.5.2	Interface de definição dos dados no sistema .....	127
5.5.2.1	Definição da base de dados ..	127
5.5.2.2	Definição de esquemas externos ...	131
5.5.3	Aspectos funcionais do embutimento ...	132
5.5.3.1	Comunicação usuário com o Banco de Dados .....	132
5.5.3.2	Conversões .....	138
5.5.3.3	Estruturas de controle de fluxo ..	138
5.5.3.4	Considerações sobre proteção dos dados .....	139

5.5.3.5	Considerações sobre homogeneidade	140
5.5.4	Aspectos de implementação do embutimento ...	141
5.5.4.1	Descrição da forma de embutimento	141
5.5.4.2	Considerações sobre a portabilidade da linguagem .....	143
5.6	O Sistema DMS II .....	144
5.6.1	Caracterização do sistema .....	144
5.6.1.1	Introdução .....	144
5.6.1.2	Estruturas de dados .....	144
5.6.1.3	Linguagens hospedeiras .....	145
5.6.1.4	Estrutura funcional .....	145
5.6.2	Interface de definição dos dados no sistema	147
5.6.2.1	Definição da base de dados ..	147
5.6.2.2	Definição de esquemas externos	152
5.6.3	Aspectos funcionais do embutimento ...	153
5.6.3.1	Comunicação usuário com o Banco de Dados .....	153
5.6.3.2	Conversões .....	160
5.6.3.3	Estruturas de controle de fluxo ..	160
5.6.3.4	Considerações sobre proteção dos dados .....	160
5.6.3.5	Considerações sobre homogeneidade	161
5.6.4	Aspectos de implementação do embutimento	162
5.6.4.1	Descrição da forma de embutimento	162
5.6.4.2	Considerações sobre a portabilidade da linguagem .....	165
5.7	O Sistema INGRES .....	166
5.7.1	Caracterização do sistema .....	166
5.7.1.1	Introdução .....	166
5.7.1.2	Estruturas de dados .....	166
5.7.1.3	Linguagens hospedeiras .....	167
5.7.1.4	Estrutura funcional .....	167
5.7.2	Interface de definição dos dados no sistema	168
5.7.2.1	Definição da base de dados ..	168
5.7.2.2	Definição de esquemas externos ..	170
5.7.3	Aspectos funcionais do embutimento ...	171



5.7.3.1	Comunicação usuário com o Banco de Dados .....	171
5.7.3.2	Conversões .....	176
5.7.3.3	Estruturas de controle de fluxo ..	177
5.7.3.4	Considerações sobre proteção dos dados .....	178
5.7.3.5	Considerações sobre homogeneidade	179
5.7.4	Aspectos de implementação do embutimento	180
5.7.4.1	Descrição da forma de embutimento	180
5.7.4.2	Considerações sobre a portabilidade da linguagem .....	182
5.8	O Sistema R .....	183
5.8.1	Caracterização do sistema .....	183
5.8.1.1	Introdução .....	183
5.8.1.2	Estruturas de dados .....	183
5.8.1.3	Linguagens hospedeiras .....	184
5.8.1.4	Estrutura funcional .....	184
5.8.2	Interface de definição dos dados no sistema	185
5.8.2.1	Definição da base de dados ..	185
5.8.2.2	Definição de esquemas externos ...	186
5.8.3	Aspectos funcionais do embutimento ...	187
5.8.3.1	Comunicação usuário com o Banco de Dados .....	187
5.8.3.2	Conversões .....	197
5.8.3.3	Estruturas de controle de fluxo ..	197
5.8.3.4	Considerações sobre proteção dos dados .....	198
5.8.3.5	Considerações sobre homogeneidade	201
5.8.4	Aspectos de implementação do embutimento	202
5.8.4.1	Descrição da forma de embutimento	202
5.8.4.2	Considerações sobre a portabilidade da linguagem .....	206
5.9	PASCAL/R .....	208
5.9.1	Introdução .....	208
5.9.2	Estruturas de dados .....	209
5.9.2.1	Retrospectiva sobre as construções PASCAL .....	209

5.9.2.2	Novas construções implementadas ..	211
5.9.3	Definição dos dados .....	211
5.9.3.1	Estruturas tipo relação .....	211
5.9.3.2	Estruturas tipo Banco de Dados ..	212
5.9.4	Manipulação dos dados .....	213
5.9.4.1	Uso de expressões no PASCAL/R	214
5.9.4.2	Instruções de manipulação de relações como um todo .....	216
5.9.4.3	Instruções de manipulação de rela ções no modo "uma-tupla-por-vez "	219
5.9.5	Considerações sobre proteção dos dados ...	220
5.9.6	Estruturas de controle de fluxo .....	220
5.9.7	Considerações finais .....	222
6.	COMPARAÇÃO E CLASSIFICAÇÃO DAS CARACTERÍSTICAS DA DML NAS DIVERSAS FORMAS DE EMBUTIMENTO .....	223
6.1	Considerações preliminares .....	223
6.2	Comparação a nível de dados .....	224
6.2.1	Como o usuário define a área de comuni cação de dados .....	225
6.2.2	Criação dinâmica de estruturas do B.D. a nível de programa .....	230
6.2.3	Conversão de tipos de dados .....	233
6.3	Comparação a nível de instruções .....	234
6.3.1	Homogeneidade .....	235
6.3.2	Poder de seleção .....	239
6.3.3	Estruturas de controle de fluxo .....	248
6.3.4	Proteção dos dados .....	250
6.4	Comparação a nível de mensagens .....	256
6.4.1	Tratamento de exceções .....	257
6.5	Quadro sintético de comparação das caracte rísticas .....	259
6.6	Conclusões delineadas sobre a comparação rea lizada .....	261
7.	ESTUDO DE CASO: ENQUADRAMENTO DE LOBAN NA CLASSIFI CAÇÃO PROJETADA .....	270
7.1	Considerações preliminares .....	270

7.2	Descrição de LOBAN no contexto da classificação .....	271
7.2.1	Caracterização do sistema .....	271
7.2.1.1	Estruturas de dados .....	271
7.2.1.2	Linguagens hospedeiras .....	271
7.2.1.3	Estrutura funcional .....	272
7.2.2	Interface de definição dos dados no sistema .....	273
7.2.2.1	Definição da base de dados ..	273
7.2.2.2	Definição de esquemas externos	274
7.2.3	Aspectos funcionais do embutimento ...	274
7.2.3.1	Comunicação do usuário com o Banco de Dados .....	274
7.2.3.2	Conversões .....	280
7.2.3.3	Estruturas de controle de fluxo .....	280
7.2.3.4	Considerações sobre proteção dos dados .....	281
7.2.3.5	Considerações sobre homogeneidade .....	282
7.2.4	Aspectos de implementação do embutimento .....	282
7.2.4.1	Descrição da forma de embutimento .....	282
7.2.4.2	Considerações sobre a portabilidade da linguagem .....	283
7.3	Avaliação das características de LOBAN no contexto da classificação .....	285
8.	CONCLUSÕES E SUGESTÕES .....	292
9.	BIBLIOGRAFIA .....	294

## LISTA DE ABREVIATURAS

ABD	Administrador do Banco de Dados
BD	Banco de Dados
DDL	Linguagem de Definição de Dados ("Data Definition Language")
DML	Linguagem de Manipulação de Dados ("Data Manipulation Language")
S.G.B.D.	Sistema de Gerência de Banco de Dados

## LISTA DE FIGURAS

FIGURA 2.1	Arquitetura funcional de um S.G.B.D. ...	21
FIGURA 3.1	Comunicação entre programa do usuário e o S.G.B.D. ....	27
FIGURA 5.1	Banco de dados exemplo .....	53
FIGURA 5.2	Arquitetura funcional do TOTAL .....	59
FIGURA 5.3	Canais de comunicação programa de aplicação e TOTAL .....	65
FIGURA 5.4	Quadro de instruções DML do TOTAL .....	71
FIGURA 5.5	Arquitetura funcional do ADABAS .....	80
FIGURA 5.6	Canais de comunicação programas de aplicação e ADABAS .....	86
FIGURA 5.7	Quadro de instruções DML do ADABAS .....	93
FIGURA 5.8	Arquitetura funcional do IMS .....	102
FIGURA 5.9	Canais de comunicação programas de aplicação e IMS .....	109
FIGURA 5.10a	Ordem de pesquisa nos tipos de segmentos	114
FIGURA 5.10b	Ordem de pesquisa em ocorrências de segmentos .....	114
FIGURA 5.11	Quadro de instruções DML do IMS .....	117
FIGURA 5.12	O IMS embutido em linguagem hospedeira .	122
FIGURA 5.13	Arquitetura funcional do IDMS .....	126
FIGURA 5.14	Canais de comunicação programas de aplicação e IDMS .....	133
FIGURA 5.15	Quadro de instruções DML do IDMS .....	137
FIGURA 5.16	O IDMS embutido em linguagem hospedeira	142
FIGURA 5.17	Arquitetura funcional do DMS II .....	146
FIGURA 5.18	Canais de comunicação programas de aplicação e DMS II .....	154
FIGURA 5.19	Quadro de instruções DML do DMS II .....	158
FIGURA 5.20	O DMS II embutido em linguagem hospedeira .....	164
FIGURA 5.21	Arquitetura funcional do INGRES .....	168
FIGURA 5.22	Canais de comunicação programas de aplicação e INGRES .....	172
FIGURA 5.23	Estrutura do processo INGRES .....	180

FIGURA 5.24	Arquitetura funcional do SISTEMA R .....	185
FIGURA 5.25	Canais de comunicação programas de aplicação e SISTEMA R .....	188
FIGURA 5.26	Quadro de instruções DML do SQL .....	196
FIGURA 5.27	Passos da precompilação no SQL .....	203
FIGURA 5.28	Passos da execução no SQL .....	204
FIGURA 6.1	Quadro sintético de comparação de características .....	260
FIGURA 7.1	Arquitetura funcional do Sistema L .....	272
FIGURA 7.2	Canais de comunicação programas de aplicação e Sistema L .....	275
FIGURA 7.3	Processo de tradução de LOBAN p/PASCAL .	284
FIGURA 7.4	A linguagem LOBAN no quadro sintético de comparação das características .....	291

## RESUMO

Este trabalho visa estabelecer uma análise comparativa das formas e métodos de embutimento de linguagens de manipulação de banco de dados em linguagens convencionais de programação a fim de que possa ser usada como ferramenta de auxílio ao desenvolvimento de futuros projetos nesta área.

Este estudo procurou analisar as preocupações existentes no projeto do embutimento. Paralelamente foram abordadas alternativas para interface com um banco de dados, considerando os aspectos humanos envolvidos, a fim de que fosse possível extrair alguns parâmetros de comparação.

Uma análise foi feita, sobre os S.G.B.D. mais expressivos em cada uma das abordagens de estruturação de dados, buscando fornecer os subsídios necessários à elaboração da comparação e classificação das características da linguagem de manipulação de dados, nas diversas formas de embutimento, a qual é considerada como objetivo principal desta dissertação.

## ABSTRACT

This work intends to establish a comparative analysis of the forms and methods of embedding database manipulation languages in general purpose programming languages, so that it can be used as a tool to assist in the future development in this area.

This study attempted to analyse the apprehensions that appears in embedding designs. Concomitantly, some other alternatives, considering the human factors involved, were approached for interfacing with a database, in order to be able to obtain some parameters of comparison.

It was done an analysis, about the most expressive D.B.M.S. for every approach of data modeling, trying to find a way to provide the necessary resources for the improvement of the comparison and classification of data manipulation languages features, in the various embedding forms, which is considered as the main purpose of this dissertation.



## 1. INTRODUÇÃO

Nos últimos anos, inúmeras linguagens de manipulação de dados e consulta a banco de dados têm sido propostas. Estudos intensos são realizados para, cada vez mais, facilitar a manipulação e manutenção das informações armazenadas no banco de dados e aumentar a flexibilidade e portabilidade das linguagens que gerenciam e manipulam estas informações.

Os projetos de sistemas de informações propostos e desenvolvidos recentemente procuram, sempre que for possível, considerar, como premissa principal, que a interface do sistema com o usuário da aplicação utilize uma comunicação que se aproxime ao máximo da forma natural de expressão e compreensão por parte deste usuário (linguagem orientada ao homem).

Muitas das tarefas atuais em interfaces de S.G. B.D. têm o usuário casual em mente e iniciam de uma análise ou percepção de suas necessidades. O resultado tem sido um grande número de linguagens que são principalmente orientadas para comunicação interativa.

Apesar da proliferação de linguagens de banco de dados para usuários casuais, integrar a interface de um determinado sistema de gerência de banco de dados em uma linguagem de alto nível permanece sendo uma importante tarefa. Linguagens modernas de programação com facilidades para declaração de tipo e uma bem desenvolvida concepção de módulo podem facilitar consideravelmente a tarefa de embutimento de tais interfaces pelos projetistas de sistemas.

Verifica-se, também, que, no uso de sistemas de gerência de banco de dados em aplicações particulares, os usuários freqüentemente necessitam de recursos adicionais, além das consultas ao banco de dados ou incessantes inclusões de novos dados. Ocasionalmente, eles esperam extrair

novas e vantajosas informações desse banco de dados pela condensação do dado em unidades mais significativas de acordo com o uso de mais ou menos algoritmos. O S.G.B.D., normalmente, oferece pouca ajuda em determinadas rotinas como somas, médias, etc., ficando a programação destas funções sob responsabilidade do usuário.

Adicionalmente, um problema existente no projeto de um S.G.B.D. é a forma de conduzir a saída das informações solicitadas. Para que os dados obtidos possam ser analisados e manipulados, além de apenas mostrados, é necessário prover estas linguagens de banco de dados dos recursos que permitam, por exemplo, formatação de dados para um relatório, avaliação de expressões aritméticas e lógicas e manipulação de seqüências de caracteres. Geralmente estas funções são exercidas por instruções de programação tradicionais, através do embutimento da linguagem de banco de dados em uma linguagem de programação (chamada, então, linguagem hospedeira).

Torna-se tarefa do projetista, do acoplamento da DML a uma linguagem convencional de programação, compatibilizar as construções de ambas as linguagens, com base nas características de cada uma.

A forma como é feito o embutimento de uma linguagem de banco de dados em linguagem hospedeira tem variações, trazendo diferenças no que diz respeito à flexibilidade para definição e manipulação de dados, independência de dados, proteção, tratamento de exceções, etc.

Percebe-se que muita ênfase está sendo dada ao problema de conciliar facilidades de manipulação de dados proporcionadas pelos S.G.B.D. com as facilidades de processamento da informação, normalmente disponíveis nas linguagens de programação. Projetos e implementações de embutimentos de DML's em linguagens hospedeiras estão cada vez se difundindo mais, considerando as diversas combinações de abor-

dagens de S.G.B.D. (relacional, em redes, hierárquico) com as linguagens de programação mesmo que estas sejam de cunho científico, como por exemplo o FORTRAN [/BAG 79/,/STA 74a/].

Logicamente, cada forma e método de embutimento terá as suas peculiaridades que vão depender, principalmente, das características das linguagens que serão acopladas.

Com base nas asserções aqui expostas, foi desenvolvida a presente dissertação. O trabalho procurou, inicialmente, focalizar a interface de manipulação de dados no contexto da arquitetura de um S.G.B.D. e analisar as preocupações que deverão ser consideradas no projeto do embutimento de linguagens de manipulação de banco de dados em linguagens convencionais de programação.

Foi realizado, também, um estudo de alternativas para interface com o banco de dados levando em conta os aspectos humanos relacionados. Após, foi realizada uma análise de linguagens para manipulação de banco de dados na qual foram considerados os S.G.B.D. mais expressivos em cada abordagem de estruturação dos dados. Esta análise trouxe os subsídios necessários para efetivar uma comparação e classificação das características da DML nas diversas formas de embutimento que foi um dos principais objetivos desta dissertação.

Para complementar a comparação e classificação resultante, foi realizado um estudo sobre o protótipo da linguagem LOBAN, numa tentativa de enquadramento desta linguagem, no contexto da classificação projetada, a fim de permitir um refinamento sobre as conclusões oriundas do quadro de comparação e classificação da DML nas diversas formas de embutimento.

Esta dissertação pretende organizar sistematicamente os tópicos relevantes no projeto de embutimento de DML's em linguagens hospedeiras através de uma comparação e

classificação das características das DML's em cada uma das formas de embutimento (taxonomia de embutimentos), esperando que este trabalho possa ser utilizado como uma ferramenta de auxílio a projetos futuros de linguagens para manipulação de banco de dados.

## 2. INTERFACE DE MANIPULAÇÃO DE DADOS NO CONTEXTO DA ARQUITETURA DE UM S.G.B.D.

A arquitetura funcional de um sistema de gerência de banco de dados poderia ser formalizada como mostra a figura 2.1.

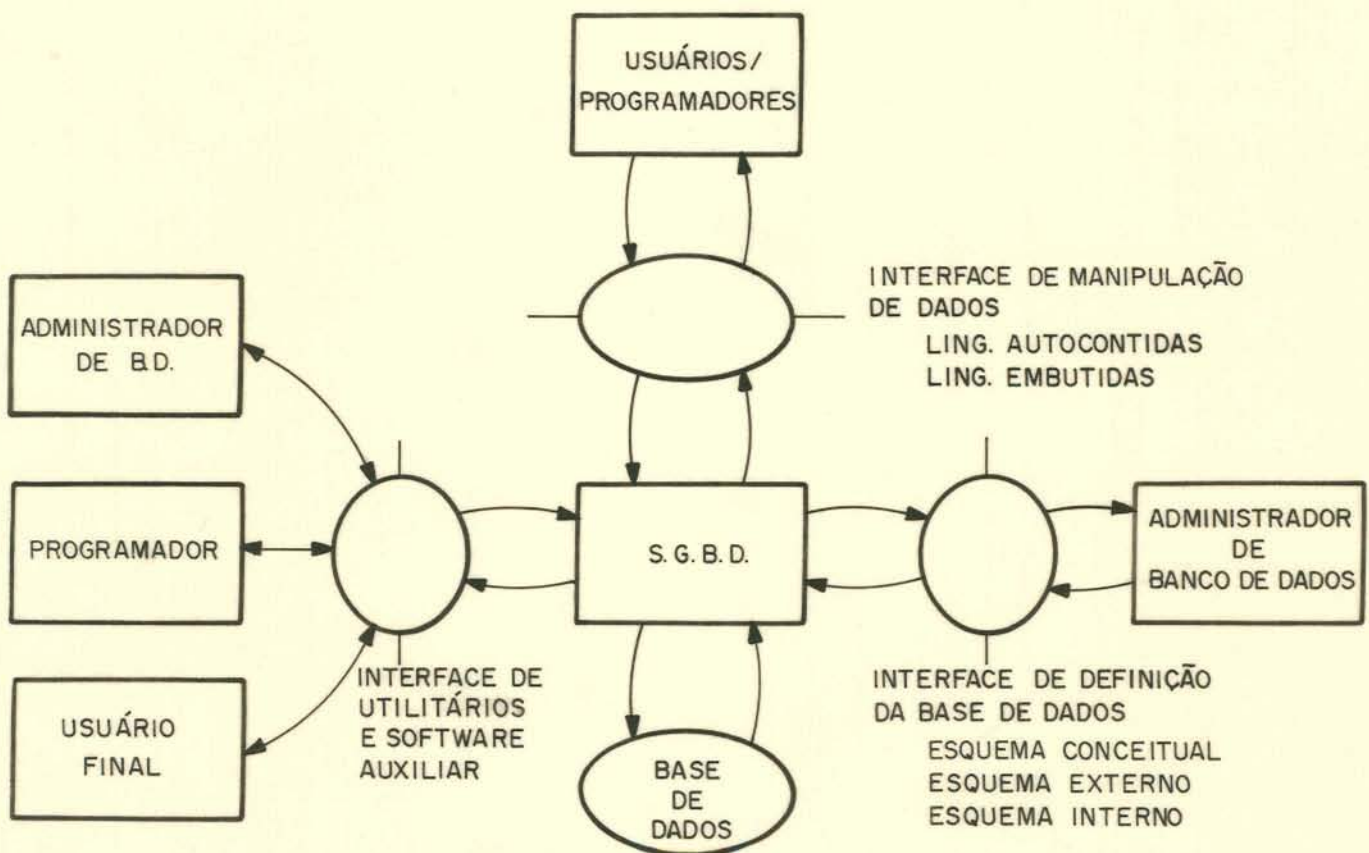


Figura 2.1 - Arquitetura funcional de um S.G.B.D.

Analisando a figura e generalizando a arquitetura, constata-se que três são as interfaces que se salientam:

- . Interface de definição da base de dados
- . Interface de manipulação dos dados

. Interface de utilitários e software auxiliar.

Através da interface de definição da base de dados é materializada a definição das estruturas dos dados (esquema) que serão armazenados no banco de dados. A tarefa de definição é geralmente confiada a uma autoridade central, conhecida como administrador do banco de dados.

Segundo a proposição formulada pelo grupo de estudos em S.G.D.B. da ANSI/X3/SPARC constante na literatura [ /TSI 78/ ], a definição dos dados segue uma arquitetura baseada em 3 níveis complementares (esquema conceitual, esquema externo, esquema interno).

O esquema conceitual ("CONCEPTUAL SCHEMA") contém a "visão do mundo real da empresa que está sendo modelada no banco de dados". Deve conter informações relativas a todas as entidades da empresa, assim como seus atributos, relacionamentos e restrições de acesso proporcionando, também, as bases para os procedimentos de segurança e integridade a serem estabelecidos sobre o banco de dados. Esta definição é o resultado da análise de dados, devendo ser elaborada em conjunto com o administrador da empresa ou outras pessoas "chave" da organização empresarial.

A definição a nível de esquema externo é realizada a partir do esquema conceitual e contém a definição dos dados necessários a uma classe de aplicações específicas obedecendo às características particulares dessas aplicações. É portanto a representação do banco de dados a nível de uma classe de aplicações particular, onde os programas de aplicações acessam o subconjunto do banco de dados necessário a seu funcionamento.

Já a definição a nível de esquema interno é realizada através da descrição da representação de armazenamento físico de dados. Neste esquema fica também a descrição de como os dados são armazenados.

Em muitos casos a definição da base de dados, a nível de esquema externo e esquema interno, é realizada por uma linguagem única que engloba a descrição de ambos os esquemas. Em outros casos, no entanto, um sistema de gerência de banco de dados é concebido usando-se, para a definição dos dados, duas ou mais linguagens de descrição que concretizam o esquema externo e o esquema interno.

Na interface de manipulação de dados encontram-se as facilidades que permitem a recuperação, inclusão, exclusão e alteração de dados da base de dados. Esta interface, conhecida também por DML ("data manipulation language") pode ser implementada através de dois enfoques distintos, ou seja, através de uma linguagem autocontida ou através de linguagens embutidas em linguagens de programação convencionais, sendo estas últimas consideradas linguagens hospedeiras.

As linguagens autocontidas possuem todas as facilidades de manipulação de dados armazenados no banco de dados. Estas linguagens precisam conter, além das facilidades citadas, recursos adicionais que permitam a formatação dos resultados obtidos a fim de que possam ser interpretados pelo usuário. Precisam, também, possuir em seu rol de construções, algumas operações adicionais de entrada de dados e lógica interna de programa que comumente são encontradas em linguagens convencionais de programação.

Em contrapartida existe a opção de implementar a interface de manipulação de dados através do uso de linguagens hospedeiras, nas quais estarão embutidas instruções específicas que permitirão a manipulação das informações mantidas no banco de dados. Com esta alternativa, o projetista da interface fica resguardado da tarefa de incluir nesta interface as operações de entrada e saída e lógica interna de programa pois estas já estarão à sua disposição pelas próprias construções da linguagem hospedeira.

As linguagens autocontidas geralmente são projetadas para atender, principalmente, as necessidades de um usuário casual sem nenhum conhecimento específico nesta área ou com algum conhecimento mas sem nenhuma especialização. Contudo, estas linguagens são também muito usadas pelos programadores de aplicação em suas tarefas de depuração dos programas. Por outro lado, pode-se salientar que, para o caso da interface de manipulação de dados baseada em linguagens hospedeiras, a comunidade de usuários que fazem uso desta alternativa se restringe aos programadores de aplicação.

A interface de utilitários, por sua vez, cumpre as funções de auxílio a administração operacional do banco de dados para cópia, reconstrução, reorganização, inicialização e carga; apoio ao administrador de banco de dados na manutenção do sistema; e auxílio ao desenvolvimento de aplicações como geradores de relatórios e pacotes de recuperação de informações.

Embora a interface de utilitários seja significativa numa arquitetura de sistemas de gerência de banco de dados, o papel das linguagens de banco de dados aparece de forma expressiva nas outras 2 interfaces.

Quanto ao aspecto da comunidade de usuários que tem contato com esta interface pode-se considerar que, dependendo do tipo de utilitário ou software auxiliar, farão uso deste aplicativo o administrador do banco de dados, o programador de aplicações e também o próprio usuário final.

Pode-se concluir, desta análise de interfaces, que o embutimento é usado, particularmente, na interface de manipulação de dados embora existam certos S.G.B.D. que admitam também uma definição dinâmica de dados, através dos programas de aplicação, necessitando, com isto, que estas construções, também, sejam embutidas na linguagem hospedeira e que uma inteligência maior seja atribuída também a nível das interfaces.



### 3. EMBUTIMENTO DE LINGUAGENS DE MANIPULAÇÃO DE UM S.G.B.D.

#### 3.1 Considerações preliminares

O embutimento de uma interface do sistema de gerência de banco de dados em uma linguagem de programação de alto nível deve aparecer ao programador como uma extensão lógica da linguagem [ /BEV 80/ ].

Generalizando este embutimento pode-se considerar que uma das tarefas iniciais seria a de incluir na linguagem hospedeira construções que equivalham ou simulem cada construção da interface do sistema de gerência de banco de dados. A dificuldade desta tarefa vai depender da forma como o embutimento será implementado. São necessárias construções para a definição de dados a nível do programa, condizente com o especificado na DDL (construções para declaração de tipo), e instruções (usando as novas funções providas pela extensão da linguagem ou funções existentes na linguagem hospedeira) para executarem as operações sobre o banco de dados (DML). As novas construções devem estar de acordo com a sintaxe da linguagem hospedeira considerando que sua semântica corresponda com as construções do sistema de gerência de banco de dados.

O projeto da interface, portanto, deverá considerar os recursos necessários para transformar o padrão de definição de dados do sistema de gerência de banco de dados (DDL) para uma máscara de definição do padrão da linguagem hospedeira e para equivalente transformação das operações da DML para operadores próprios do programa (através de macros "CALL" ou comandos especiais embutidos).

Em tempo de execução, as operações sobre o banco de dados a nível da linguagem hospedeira são mapeadas para as correspondentes primitivas de acesso às estruturas do banco de dados, exigindo uma transformação inversa.

Usando as novas construções, um usuário pode declarar um esquema de banco de dados dentro da linguagem hospedeira da mesma forma como declara arrays ou registros. Ele, no entanto, não terá uma liberdade completa nesta declaração visto que o esquema do banco de dados já está disponível a nível do sistema de gerência de banco de dados e deve haver uma compatibilidade entre estas duas declarações.

### 3.2 Preocupações no projeto do embutimento

No decorrer de um projeto de embutimento de uma linguagem de manipulação de dados em uma linguagem hospedeira alguns problemas surgem e devem ser analisados. Entre estes podem ser enumerados:

- . Interface entre o sistema de banco de dados e a linguagem hospedeira
- . Homogeneidade
- . Tipos de dados e conversões
- . Estruturas de controle de fluxo (iterações)
- . Tratamento de exceções
- . Formas de implementar o embutimento.

A seguir, é apresentado com mais detalhes, cada uma das preocupações acima mencionadas.

#### 3.2.1 Interface entre o sistema de banco de dados e o ambiente da linguagem

Um programa de aplicação passa requisições ao sistema de gerência de banco de dados e recebe dados e informações de controle (mensagens). O mecanismo típico para comunicação é ter uma área de trabalho compartilhada, como pode ser visto na figura 3.1 [ /BEV 80/ ].

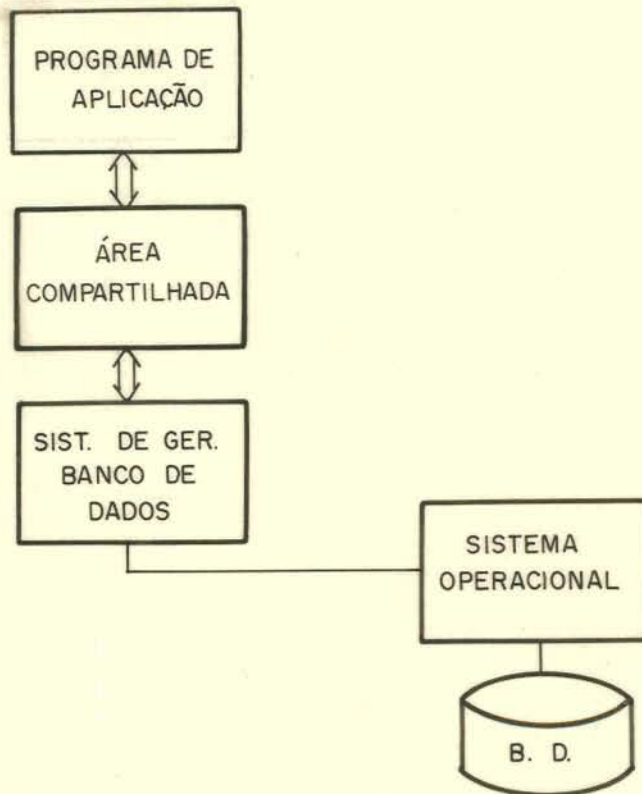


Figura 3.1 - Comunicação entre programa do usuário e o S.G.B.D.

O problema está em quanto desta área de trabalho compartilhada o programador de aplicação deve conhecer e como ela está protegida de alterações inadvertidas. Tem os que defendem a opinião [ /STO 77/ ] de que um programador deveria conhecer o mínimo possível a respeito desta área de trabalho. Deveria ser transparente a ele. Quanto mais um programador conhecer, mais facilmente ele poderá tirar vantagens disso. Vantagens muitas vezes não desejáveis. Por exemplo, um programador pode intencionalmente modificar ponteiros correntes, normalmente setados somente pelo sistema, e isto pode levar a alguma eficiência. No entanto, estes tipos de lesões causadas à especificação podem resultar em erros críticos. Na prática, tais lesões são dificilmente identificadas e isoladas.

As interfaces entre um banco de dados e seus usuários tomam a forma de linguagens cujas características dependem do tipo de uso e da comunidade de usuários que dela se utilizam.

De um modo geral é necessário uma interface com mais de uma linguagem hospedeira visto que um banco de dados tenciona servir a uma variedade de tipos de aplicação. Programadores de aplicações comerciais e científicas compartilham dados mas com diferentes necessidades de processamento. Um exemplo típico seriam as aplicações que usam a linguagem COBOL comparadas com as aplicações na linguagem FORTRAN ou ALGOL.

Se for aceito o postulado de que um número de linguagens hospedeiras precisa ter interfaces com um banco de dados, um ponto a ser considerado reside no fato de como as diferentes interfaces para diferentes linguagens hospedeiras podem estar relacionadas, tendo em vista um interesse de padronização.

Para determinar quais formas são comuns a todas as linguagens hospedeiras, é necessário analisar a interface completa entre um banco de dados e uma linguagem hospedeira a fim de ver como as funções que manipulam o banco de dados se relacionam com este e com as estruturas da linguagem hospedeira. Um ponto a ser considerado é como a descrição de dados do banco de dados (descrita pela DDL) é visualizada no programa da linguagem hospedeira. Para esta conotação, deveria-se ter um módulo de adaptação dos dados para os padrões do programa de aplicação descrevendo os dados do banco de dados em uma forma apropriada para uma determinada linguagem hospedeira e que serão utilizados pelos comandos DML embutidos nesta linguagem hospedeira. Tem-se, por exemplo, linguagens de programação (como o ALGOL) que não detêm em sua estrutura o conceito de registro e, nesse caso, o módulo acima mencionado deverá valer-se de recursos alternativos, como arrays, para materializar a conversão para os

padrões desta linguagem. Deverá, portanto, ser proporcionada uma conversão de padrões e um mapeamento adequado para tratamento das diferenças nas estruturas [/STA 74b/].

O projetista deverá procurar sempre escolher estruturas de conversão que levem a uma padronização entre as diferentes linguagens hospedeiras, alvos de um embutimento, tendo em vista que isto facilitará a compreensão do projeto e, também, a tarefa de manutenção.

No projeto da interface, especial atenção deverá ser dada aos eventuais problemas de conflitos que podem ocorrer entre linguagem de manipulação de dados e linguagem hospedeira. Encontra-se este tipo de conflito, por exemplo, na implementação de embutimento de linguagens de manipulação de S.G.B.D. relacionais ou em rede em linguagens de programação convencionais que, normalmente, seguem um estilo hierárquico em sua estrutura. Outro conflito pode aparecer, por exemplo, na abordagem tipo CODASYL, baseando sua descrição de dados em registros em contraste com a abordagem relacional onde os itens de dados elementares devem, individualmente, serem transferidos para variáveis do programa de aplicação. A preocupação com este conflito se materializa quando as duas abordagens estiverem simultaneamente presentes em uma mesma linguagem hospedeira podendo, inclusive, repercutir no aspecto da dificuldade de aprendizagem das novas construções na linguagem.

Com base no que foi argumentado aqui, pode-se concluir o seguinte:

. A descrição dos dados usados para referenciar o banco de dados a nível de programa de aplicação por definição é dependente da linguagem hospedeira.

. Funcionalmente a DML é independente da linguagem hospedeira ao passo que todas as dependências na representação no ambiente da linguagem hospedeira são manipuladas através da descrição dos dados no programa.

. Deverá ser proporcionada uma DML para cada linguagem hospedeira em particular, numa forma orientada a sintaxe destas linguagens.

### 3.2.2 Homogeneidade

Para abordar este tópico, deverão ser considerados os aspectos relacionados com duas formas de se analisar uma uniformidade no embutimento de novas construções na linguagem hospedeira comparado com as construções existentes no sistema de gerência de banco de dados. Estas formas poderiam ser classificadas em:

- . Homogeneidade DDL X definição no programa
- . Homogeneidade DML X comandos padrões do programa.

As duas formas de caracterizar a uniformidade têm uma influência significativa nos aspectos humanos que envolvem o projeto de uma linguagem, principalmente pelo fato de que quanto mais homogêneo se apresentar o acoplamento entre as linguagens, menos dispendiosa será a tarefa de treinamento do usuário nas novas construções da linguagem e, por conseqüente, a revisão na metodologia de programação será aceita de forma mais natural.

Quando foi comentado, anteriormente, o problema da interface entre o sistema de gerência de banco de dados e a linguagem hospedeira, colocou-se em discussão a questão da padronização entre as interfaces das diversas linguagens hospedeiras e a das incumbências do módulo de adaptação das estruturas e declarações. Pode-se salientar, agora, que os problemas pertinentes a estas questões são amenizados se existir uma homogeneidade significativa entre a linguagem de definição dos dados (DDL) e a forma como os dados são declarados no programa de aplicação.

Para a segunda forma de homogeneidade, por sua

vez, percebe-se reflexos mais diretos sobre a disciplina de programação visto que, quanto mais uniforme forem os comandos de manipulação (DML) comparados, em suas características sintáticas e semânticas, com os comandos padrões da linguagem hospedeira, mais facilmente serão manipulados pelo programador e menos repercussão sobre o ambiente de programação terá a implementação do embutimento.

Convém salientar, porém, que apesar do projetista procurar sempre ter, como meta, alcançar uma uniformidade nas linguagens que irão ser acopladas, o programador deverá sempre ter em mente que ele está trabalhando, simultaneamente, com dois ambientes diferentes e o resultado harmonioso deste acoplamento de linguagens vai, basicamente, depender do uso adequado das construções da linguagem hospedeira no contexto do banco de dados, fator este que está associado com uma disciplina de programação ou metodologia de desenvolvimento de programas. De nada valerá o esforço do projetista em procurar evitar possíveis conflitos entre as 2 linguagens se, posteriormente, o programador adotar uma metodologia de programação que não condiz com uma disciplina normalmente adotada para linguagens de manipulação de dados. Novas estruturas são anexadas às já existentes na linguagem hospedeira e esta preocupação deve ser considerada pelo programador quando este for adequar a sua disciplina de programação à nova situação.

### 3.2.3 Tipos de dados e conversões

Seria muito interessante que o conjunto de tipos primitivos, como por exemplo seqüência de caracteres, inteiros e reais, disponíveis no sistema de gerência de banco de dados e no ambiente da linguagem hospedeira, fossem os mesmos. Se isto não ocorrer, depara-se com um problema óbvio de conversão. Para ressaltar o fato, um exemplo ilustrativo seria o caso da especificação de um tipo de dado como "dias

da semana" no ambiente do banco de dados. Se este tipo de dado for tratado como uma simples seqüência de caracteres em um programa de aplicação, informações ilegais podem aparecer.

Outro problema ocorre quando um dado de um determinado tipo no banco de dados é transferido para uma variável do programa declarada como sendo de outro tipo (ex.: de cimal p/inteiro).

A conversão entre tipos de dados pode ser implícita ao sistema ou ser realizada explicitamente através de uma especificação por parte do programador de aplicação ou do administrador do banco de dados.

A conversão é um problema complexo porque ela tem influência na integridade, complexidade e eficiência do programa. Propiciando ao usuário, programador ou administrador do banco de dados, controle explícito sobre conversões, será proporcionado, em parte, maior proteção contra conversões não intencionais. Contudo, exigindo especificação explícita da conversão, significa que programas simples, envolvendo conversões óbvias (inteiro p/real, por exemplo) tornar-se-ão mais complicados.

Por outro lado, conversão implícita pode degradar o desempenho assim como permitir que programas acessem dados numa maneira que pode não ter sido pretendida pelo projetista do banco de dados.

O problema de conversão de tipo tem sido muito argumentado. A tendência atual parece ir de encontro a um modelo forte ("STRONG TYPING") com conversões implícitas limitadas aos tipos primitivos mais simples e com o programador tendo controle explícito sobre outras conversões [STO 77/].

O problema da conversão torna-se realmente crítico quando não existir nenhuma estrutura ou tipo de dado semelhante na linguagem hospedeira que permita a realização



de uma adaptação adequada do formato do dado compartilhado pelo sistema de gerência de banco de dados e pelo programa de aplicação. Com isto o projetista terá que lançar mão de artifícios que, muitas vezes, não são muito confiáveis ou e ficientes.

#### 3.2.4 Estruturas de controle de fluxo

No projeto do embutimento de uma linguagem de banco de dados em uma linguagem hospedeira, o projetista se defronta com o problema de questionar quais as estruturas de controle de fluxo estão disponíveis para acesso aos registros do banco de dados.

Atualmente fomenta-se uma tendência conduzida em direção a asserções que salientam o fato de que construções de linguagens concisas, ou seja, linguagens que necessitem de uma quantidade mínima de código para expressar uma requisição, e também de fácil compreensão devam ser proporcionadas para estruturas de controle comumente utilizadas, como no caso de iterações ("LOOPS") [/STO 77/].

Em linguagens consideradas como sendo de características procedurais, normalmente, as iterações devem ser codificadas através de carga explícita de registros e o controle de fluxo se completa através de um teste de um código de estado retornado ao programa de aplicação pela interface com o sistema de gerência de banco de dados. Com isto, aparentemente, os programas se tornam mais complexos e confusos visto que as especificações de consultas são ocultadas no meio das demais estruturas do programa.

No caso de algumas linguagens relacionais este problema é amenizado pois a especificação de consulta iterativa é centralizada no comando de definição de um cursor atuando sobre um conjunto de tuplas. Apesar disso usando estas linguagens na modalidade de embutimento em linguagens

hospedeiras o programador deverá, também, proporcionar um controle de fluxo explícito com teste de código de estado a fim de realizar a carga de cada tupla (registro) com base neste cursor. Este controle explícito de fluxo deverá ser realizado porque o programador não sabe, de antemão, quantas tuplas foram retornadas na chamada da especificação da consulta. Contudo, do ponto de vista da operação de consulta ao banco de dados, pode-se afirmar que a especificação, com seus critérios de seleção e qualificação, ficam centralizados em um ponto do programa. Apenas um tratamento especial deverá ser dado para adaptar o resultado aos padrões e disciplina de programação do programa de aplicação.

### 3.2.5 Tratamento de exceções

Aliado ao problema das estruturas de controle de fluxo que estão disponíveis na linguagem e também aos aspectos relacionados à concisão da linguagem, em termos de quantas construções são necessárias para expressar uma requisição, estão as facilidades de tratamento de exceções de operações sobre o banco de dados refletidas na linguagem hospedeira.

Na maioria dos acoplamentos de sistemas de gerência de banco de dados a linguagens de programação convencionais, existe uma dependência das atitudes do programador em especificar, explicitamente, um teste de um código de estado, retornado da interface com o banco de dados, para verificar o sucesso ou não da operação realizada.

Por outro lado, algumas linguagens hospedeiras têm implementado facilidades de manipulação de erros produzidos em operações com o banco de dados através de um tratamento de exceções materializado mediante uma sinalização de exceções quando do uso de uma cláusula específica (ON EXCEPTION) associada a cada comando. Através destas cláusulas, o

programador especifica o procedimento que deve ser realizado em caso de ocorrer a exceção como, por exemplo, executar uma rotina de recuperação da falha, sendo que esta rotina pode ser predefinida e ser utilizada por todos os programas de aplicação que manipulam o referido sistema de gerência de banco de dados. Com isto os procedimentos de tratamento de exceção não fogem ao controle do programador apesar de não exigir, deste programador, a especificação de um teste do código de estado retornado, para verificação do sucesso da operação, em cada iteração do fluxo de recuperação dos registros do banco de dados.

### 3.2.6 Formas de implementar o embutimento

O acoplamento de uma linguagem de programação ao sistema de gerência de banco de dados pode ser realizado de três formas distintas, a saber:

- . Definindo subrotinas que executam as requisições ao banco de dados quando chamadas;

- . Embutindo construções do banco de dados em uma linguagem existente e usando um pré-processador para extrair estas construções do programa fonte e traduzí-las em chamadas ao sistema de gerência de banco de dados;

- . Designando uma nova linguagem de programação ou modificando uma já existente na qual facilidades de banco de dados são integradas no ambiente da linguagem.

Estas três formas serão discutidas e analisadas individualmente, com maiores detalhes, a seguir.

a) Embutimento via chamada a subrotina (interpretação)

Neste embutimento a interface do sistema de gerên

cia de banco de dados é erguida na linguagem de programação em uma forma essencialmente não modificada.

Basicamente, uma ou mais rotinas (procedures) são proporcionadas, sendo chamadas com uma seqüência de caracteres (string) representando uma instrução do sistema de gerência de banco de dados.

Transformações de padrão de definição de dados do sistema de gerência de banco de dados para uma máscara de definição no padrão da linguagem hospedeira são virtualmente não existentes e nenhuma verificação em tempo de compilação é feita. Cabe ao programador especificar a sua área de trabalho onde manipulará os dados recuperados do banco de dados e fica sob sua responsabilidade a definição coerente destes dados a nível do ambiente do programa de aplicação.

Pode-se considerar como vantagens desta abordagem:

- . É fácil e rápido de implementar;
- . Implementação necessita relativamente pouca complexidade;
- . Permite o uso de interfaces funcionais em linguagens orientadas a "procedures", ou seja, linguagens procedurais;
- . É flexível no que diz respeito às abordagens de dados dos sistemas de gerência de banco de dados e às linguagens de programação que estão sendo interconectadas, visto que nenhuma construção mais complexa é exigida.

Como defeitos desta abordagem, poder-se-ia salientar:

- . O usuário precisa estar completamente familiarizado com a interface de sistema de gerência de banco de dados, que ele deverá simultaneamente tratar com duas lingua-

gens;

. Erros sintáticos e lógicos no programa, relativos a interface, são raramente descobertos antes do tempo de execução.

b) Embutimento via adaptação de linguagem (compilação)

Nesta abordagem, as transformações de padrões de definição de dados do sistema de gerência de banco de dados para a linguagem hospedeira e também as transformações das operações da DML para operadores próprios do programa entram no projeto da linguagem e conseqüentemente tornam-se parte do compilador da linguagem de programação.

As linguagens procedurais são usualmente concebidas para interface com um banco de dados por meio desta abordagem. A linguagem é estendida para permitir que aloje um conjunto de comandos (DML) que proporcionam a capacidade de armazenar e recuperar dados do banco de dados.

Esta abordagem tem o mais forte de sua eficácia onde a anterior (via chamada a subrotina) tem sua deficiência e vice-versa.

Podem ser consideradas vantagens desta abordagem:

. Os tipos do sistema de gerência de banco de dados tornam-se uma parte "natural" da linguagem de programação e são adaptados a suas características;

. O código objeto é eficiente porque muitas das transformações de operadores da linguagem hospedeira para correspondentes operações DML, que executam as requisições solicitadas sobre o banco de dados, podem ser feitas durante o tempo de compilação, ou seja, parte da ligação entre os módulos pode ser realizado durante a compilação;

. A correção do programa a nível de tipos de da-

dos pode já ser estabelecido durante a compilação.

Como deficiência desta abordagem, pode-se enumerar:

- . Completa inspeção e considerável reescrita de compiladores existentes, necessitando complexidade na tecnologia de compiladores;

- . Modificações devem ser repetidas para cada linguagem de programação que utilizará a interface com o sistema de gerência de banco de dados;

- . Esforços de padronização serão necessários.

#### c) Embutimento via precompilação (filtro)

Para esta forma de embutimento, as instruções específicas do sistema de gerência de banco de dados, que precisam ser reconhecidas como tal, são preprocessadas e traduzidas em instruções próprias da linguagem hospedeira, usualmente em chamadas de subrotinas.

A precompilação pode ser considerada como um meio termo entre as duas abordagens anteriores. Sendo assim apresenta algumas das vantagens das outras duas, tal como o de proporcionar uma linguagem simples, coerente para o usuário, realizando alguns processamentos em tempo de precompilação e necessitando menos esforços de implementação.

Combina, também, algumas deficiências como:

- . A porção do sistema de gerência de banco de dados na linguagem precisa ser projetada para cada linguagem individualmente.

- . Complexidade na tecnologia de compiladores é necessária visto que, durante a precompilação, algumas tarefas, normalmente atribuídas a um compilador e que requerem conhecimento sobre esta tecnologia, devem ser executadas.

#### 4. ESTUDO DE ALTERNATIVAS PARA INTERFACE COM BANCO DE DADOS BASEADA EM ASPECTOS HUMANOS

##### 4.1 Considerações preliminares

Com o desenvolvimento das áreas de linguagens de interface do usuário com os sistemas de gerência de banco de dados surgiu a necessidade de buscar novas ferramentas de medição de desempenho, flexibilidade e facilidade de uso. Dentre estas ferramentas surgiram as experimentações psicológicas controladas [/JOY 83/, /REI 77/, /REI 81/, /SHN 78/, /SHN 83/, /WEL 81/], sobre a comunidade de usuários que fazem uso das linguagens, procurando extrair, dos resultados destes experimentos, conclusões a respeito dos aspectos humanos que estão envolvidos, a fim de permitir uma boa aceitabilidade das linguagens.

Verificou-se também que a difundida disseminação de computadores e sistemas de informação para indivíduos não treinados tecnicamente necessitava de uma nova abordagem para o projeto e desenvolvimento de interfaces de banco de dados.

Os pesquisadores na área de linguagens de programação têm aceito a utilidade de uma abordagem mais psicologicamente orientada ao estudo do comportamento do programador ou usuário final. Pesquisas estão sendo conduzidas para um aperfeiçoamento de modelos no uso de linguagens disponíveis atualmente, bem como para sugerir técnicas de melhoria na qualidade dos programas e metodologias para desenvolvimento de futuras linguagens.

Existe um difundido reconhecimento que sistemas futuros serão comercialmente viáveis somente se a interface com o usuário esteja em harmonia com as necessidades de tarefas e a habilidade desse usuário [/SHN 78/]. A eficiência no uso dos recursos do sistema pode se tornar insignificante caso o sistema não esteja projetado para uma fusão harmo

niosa com as necessidades e habilidades de seus usuários.

A intenção não é a de esgotar, aqui, todo o assunto relativo a essa nova área de pesquisa, visto que o assunto é muito vasto. Procurar-se-á comentar alguns aspectos importantes ligados a essa área e que influenciaram a escolha de certos itens de comparação das diversas formas de embutimento de linguagens de banco de dados em linguagens convencionais (hospedeiras).

#### 4.2 Considerações sobre o projeto de interface

Existe um consenso nos princípios básicos que seriam aconselháveis no projeto e implementação de interfaces para usuários finais não especializados em processamento de dados. Estes princípios dizem respeito às seguintes premissas [/DAL 76/]:

- . A linguagem do usuário deve ser simples, de fácil uso e de fácil compreensão. Deve ser compatível (a nível léxico, pelo menos) à terminologia do usuário;

- . A linguagem do usuário deve ser suficientemente poderosa para satisfazer as necessidades do usuário final;

- . A interface precisa comportar uso interativo do sistema, em parte porque uma larga proporção de usuários necessitarão, pela natureza de suas atividades, ter acesso imediato ao sistema de gerência de banco de dados e em parte porque o diálogo usuário-sistema pode ser uma condição necessária para especificação de operações do sistema;

- . A interface deve proporcionar ao usuário final uma independência de dados a nível lógico, no sentido que o usuário é isolado da necessidade de conhecer detalhes da estrutura lógica do banco de dados estando livre para visualizar os dados de forma objetiva (visões locais), e a nível



físico, no sentido que o usuário está isolado da necessidade de conhecer detalhes da organização física e caminhos de acesso ao dado. Uma consequência da independência de dados é que modificações no sistema serão transparentes ao usuário e que processos definidos pelo usuário poderão continuar a serem executados com sucesso. A independência lógica é mais problemática de ser conseguida visto que, em geral, não é possível prever a maneira pela qual o esquema conceitual pode ser modificado. Já para a independência física a condição necessária e suficiente é que a linguagem de manipulação não referencie as estruturas de armazenamento, mas sim os esquemas conceituais ou seus esquemas externos derivados (B.D. lógico).

Pode-se considerar estas premissas como sendo parte integrante dos fatores humanos que direcionam o planejamento do projeto de uma linguagem porque delas vai depender o grau de aceitação das potencialidades da linguagem por parte do usuário que a utilizará.

#### 4.3 Questionamentos sobre aspectos humanos no projeto de uma linguagem de banco de dados

No projeto de uma linguagem de banco de dados diversas variáveis sobre os aspectos humanos devem ser consideradas [/MCD 82/, /SHN 78/]. Estas variáveis envolverão um conjunto de aplicações a serem definidas sobre o banco de dados e o perfil da comunidade de usuários agindo sobre as aplicações. Com base no estudo destas variáveis do sistema, o projetista poderá estruturar a linguagem de tal forma que esta atenda às necessidades do usuário de maneira conveniente e harmoniosa.

As variáveis que devem ser abordadas numa análise de aspectos humanos envolvidos no projeto da linguagem, poderiam ser classificadas como segue:

a) Operação sobre o banco de dados (funções)

O projetista deverá enumerar e analisar todas as possibilidades de operações que o usuário terá a sua disposição sendo que estas operações atuam sobre ítems que podem ser a nível de campo, registro, conjunto de registros, arquivos ou todo o banco de dados.

Exemplificando, ter-se-ia, entre as operações a serem consideradas, as seguintes:

- . Definição de dados (esquema e subesquema)
- . Inserção de um ou mais ítems
- . Exclusão de um ou mais ítems
- . Alteração de um ou mais ítems
- . Recuperação de informações do banco de dados (consulta)
- . Chaveamento e liberação de ítems (controle de acesso)
- . Verificação de privacidade (restrições de integridade)
- . Utilitários de uso geral.

b) Incumbências a cargo do usuário

Para que o usuário se capacite a utilizar as operações sobre o banco de dados algumas incumbências preliminares são atribuídas a este usuário. Em vista disso, o projetista deverá enfatizar a otimização dos aspectos que influenciam estas incumbências e facilitar, ao máximo, a ambientação do usuário neste sentido.

Entre estas, seria conveniente citar:

- . Estudo da sintaxe e semântica da especificação das operações;
- . Combinação de operações para possibilitar a execução de determinada tarefa e conseqüente compreensão desta

combinação;

- . Depuração de sintaxe ou semântica para corrigir erros;

- . Modificação de operações ou funções previamente definidas para adequá-las à nova situação.

Conclui-se, portanto, que, a simplicidade de tratamento destas incumbências, ou seja, a facilidade com que os usuários interagem com elas, induz na linguagem um alto grau de flexibilidade e compreensibilidade.

c) Forma como o usuário se comunica com o sistema

O usuário pode ter acesso ao sistema de gerência de banco de dados por intermédio de formas distintas. Estas formas têm características e facilidades próprias que deverão ser consideradas na análise dos aspectos humanos durante o projeto da linguagem.

A intenção do usuário com o sistema pode, por exemplo, assumir as seguintes formas:

- . Linguagem hospedeira embutindo uma nova forma sintática através da simples invocação de subprogramas ou através do uso de um precompilador para interpretação das operações sobre o banco de dados;

- . Linguagens autocontidas que proporcionam todas as facilidades para execução das operações disponíveis no banco de dados;

- . Interações de processamento direto com base em aplicações específicas através de seleção de menu, preenchimento de informações em lacunas preformatadas ou requisições parametrizadas;

- . Linguagem natural / inteligência artificial.

## d) Protocolos e abrangência de consultas

Diz respeito ao tipo de informação que resultará de uma consulta feita ao banco de dados.

Pode ser visualizada em 4 níveis:

- . Simples verificação da presença ou não de um valor de item específico;
- . Recuperação de registro único com o fornecimento de uma chave;
- . Recuperação de uma coleção de registros que satisfazem uma condição de seleção;
- . Emissão do conteúdo total do arquivo principalmente na modalidade "batch".

## e) Comunidade de usuários do banco de dados

É extremamente importante que o projetista identifique todos os tipos de usuários que farão uso do banco de dados para que possa definir uma linguagem básica que se adapte às características e exigências de cada um dos usuários. Os usuários deverão ser classificados de acordo com seus conhecimentos sobre os assuntos que envolvem a computação eletrônica de dados. Sendo assim, pode-se enumerá-los em 3 categorias distintas:

- . Usuários casuais, não treinados, sem conhecimentos sobre sintaxe de linguagens próprias de computação ou sobre a estruturação dos dados;
- . Usuários que normalmente utilizam consultas sobre o banco de dados porém não possuindo interesses maiores neste assunto e nem conhecimentos generalizados na área de computação (ex.: secretárias, etc.);
- . Usuários profissionais em banco de dados (ex.:

programadores, analistas de sistemas, engenheiros de software, etc.).

As características de uma interface do usuário devem claramente refletir as características de usuários finais, para que possam proporcionar um ambiente de interação com o sistema de gerência de banco de dados que corresponda ao estabelecimento de estilos de operação do usuário quando empregado em tarefas de processamento da informação.

A tarefa do projetista do sistema é a de construir interfaces que proporcionam ao usuário uma transição aceitável dos métodos convencionais de processamento da informação (manual ou computação convencional) para a interação com uma disciplina de sistema de gerência de banco de dados. O projetista deve proporcionar contextos interativos que utilizem construções familiares ao usuário.

#### f) Características de consultas

A característica das consultas realizadas sobre o banco de dados é outra variável que deverá ser considerada a âmbito dos aspectos humanos envolvidos no projeto da linguagem pois desta será extraído o grau de dificuldade que o usuário terá para formular uma consulta. Com base em características é possível conseguir a seguinte classificação de consultas:

- . Consulta de mapeamento simples que retorna valores de dados quando um valor de outro campo é conhecido;

- . Consulta selecionando todos os valores (registro completo) associados a um valor de chave);

- . Consulta com projeção que, no modelo relacional recupera todo um domínio (coluna) ou conjunto de domínios;

- . Consultas booleanas que usam conetivos lógicos (and/or/not);

- . Consultas com operações sobre conjuntos (intersecção, união, etc.);

- . Consultas aninhadas que envolvem mais de uma operação de consulta;

- . Consultas que usam funções especiais como `max`, `min`, `avg`;

- . Consultas que recuperam conjuntos de agrupamentos de itens resultantes (ex.: na linguagem SQL a cláusula `GROUP BY`).

g) Facilidades adicionais proporcionadas na interface

Convém salientar, também, a importância que facilidades adicionais proporcionadas na interface do sistema de gerência de banco de dados terão sobre a satisfação pessoal do usuário no uso da linguagem de consulta e na minimização das tarefas que este usuário teria que realizar. Estas facilidades induzirão a uma maior confiança do usuário nas tarefas que realizará com a ajuda da linguagem de consulta.

A título de ilustração pode-se enumerar algumas das facilidades adicionais que podem estar disponíveis na interface. Por exemplo:

- . Dicionário de dados que permite ao usuário ter uma visão global de todo o sistema com suas diversas entidades e relacionamentos que o compõem;

- . Controle da saída do resultado da consulta permitindo que o usuário realize mudanças indicando um novo dispositivo de destino;

- . Formatação do resultado da consulta conforme parâmetros especificados;

- . Apoio à incerteza do usuário na escrita de con

sultas ou a incorreções produzidas (HELP).

Estas facilidades normalmente podem ser implementadas através de utilitários e software auxiliar.

#### 4.4 - Ferramentas utilizadas para mensurar facilidades de linguagens

Para mensurar as facilidades de uso de uma linguagem, a fim de extrair conclusões sobre os aspectos humanos como uma das características da linguagem, o projetista pode valer-se de algumas ferramentas de teste que o auxiliarão no experimento [/REI 81/].

Na análise de aspectos humanos envolvidos na comunicação do usuário com o banco de dados, os métodos acadêmicos de psicologia experimental são aplicados em experimentos práticos de desenvolvimento de sistemas. Para concretizar esses experimentos, os procedimentos a serem abordados são os seguintes:

- . Definir precisamente o que deve ser mensurado;
- . Desenvolver padrões de comparação do desempenho do usuário mediante uso de funções que substituem as tarefas do usuário (tarefas do usuário artificialmente realizadas);
- . Medir parâmetros relevantes do desempenho do usuário (ex.: número de erros, tempo para interpretação de um mostrador luminoso).

Cabe ao projetista ou experimentador enfatizar os aspectos ligados a aprendizagem e compreensão que são muito significativos na mensuração das facilidades de uso das linguagens de consulta.

Num ambiente experimental podem ser adotadas as

seguintes ferramentas de mensuração:

. Escrita de consulta: ao usuário é solicitada a transcrição de uma consulta em linguagem natural para uma dada linguagem de consulta ao banco de dados;

. Leitura de consulta: inverso da anterior;

. Interpretação de consulta: é fornecido ao usuário um trecho de consulta escrita em linguagem de consulta ao banco de dados e o conteúdo armazenado neste banco de dados e é solicitado que indique quais os dados que serão recuperados;

. Compreensão da questão: é fornecido ao usuário uma questão de consulta em linguagem natural e o conteúdo armazenado no banco de dados solicitando que indique o resultado que será obtido;

. Memorização: usado para averiguar a memorização dos efeitos de uma consulta;

. Resolução de problemas: é fornecido ao usuário um banco de dados e um problema típico de aplicação sobre este banco de dados e é solicitado que o referido usuário forneça questões de consulta em linguagem natural para solucionar o problema apresentado.

Aliado a essas ferramentas o experimentador pode lançar mão de alguns testes adicionais que auxiliam na mensuração. São eles:

. Exame final da aprendizagem: verifica o quanto fácil uma linguagem de consulta é assimilada pela aprendizagem;

. Compreensão imediata: ajuda a identificar porque problemas particulares de aprendizagem ocorrem e quais os tópicos que não foram inteiramente assimilados pelo usuário. É realizado durante a aprendizagem, imediatamente após



uma função ter sido ensinada;

. Revisão: tem finalidades similares ao teste de compreensão imediata, no entanto é realizado após o processo de aprendizagem;

. Produtividade: testam o quão bem uma linguagem pode ser usada após alguns níveis de aprendizagem predeterminados terem sido alcançados;

. Retenção: está relacionado com o processo de memorização. Testa o quão fácil uma linguagem de consulta é relembrada;

. Reaprendizagem: teste de quão fácil uma linguagem de consulta é reaprendida por usuários que deixaram de utilizá-la por algum tempo.

Percebe-se que os problemas relativos aos aspectos humanos envolvidos nas características de uma linguagem de consulta se materializam, principalmente nos seguintes tópicos:

. Tempo necessário para aprendizagem da linguagem;

. Fator de desempenho da linguagem - tempo de resposta nas consultas (interação "on-line");

. Satisfação subjetiva do usuário com o uso da linguagem;

. Tratamento de erros eventuais - subsistema de ajuda ao usuário (help) para contornar falhas;

. Coeficiente de retenção do conhecimento adquirido no treinamento sobre a linguagem. Está relacionado com a facilidade da aprendizagem.

Viu-se, portanto, que o projetista tem, a seu dispor, uma série de ferramentas que podem auxiliá-lo a reali-

zar experimentos práticos, cujas conclusões, na área de fatores humanos, comparando facilidades de uso de um conjunto de linguagens de consulta, podem ser usadas como padrão para o desenvolvimento de uma nova linguagem ou adaptação das características de uma linguagem problemática.

A combinação de testes e ferramentas utilizadas para mensurar as facilidades vai depender das circunstâncias e do tipo de aplicação que terão.

#### 4.5 Breves comentários relativos a experimentos sobre linguagens

Analisando a literatura sobre o assunto de aspectos humanos envolvidos no desempenho de linguagens [/REI 81/, /REI 77/, /WEL 81/] é possível encontrar a descrição de diversos experimentos realizados para comparar as características e facilidades de linguagens de consulta a banco de dados.

Verifica-se que, com o crescente interesse na melhoria das facilidades de uso do computador, um número significativo de experimentos tem considerado o uso de técnicas de experimentos psicológicos para estudo de linguagens de programação. Nestas pesquisas as tendências mais frequentes são [/REI 77/]:

- . Estudos comparativos de linguagens específicas;
- . Análise de como as pessoas programam, pela análise de protocolos de consultas ou pelo estudo do processo de depuração dos programas;
- . Análise de erros produzidos pelos programadores;
- . Comparação de ferramentas particulares de linguagens tais como comandos IF aritméticos X lógicos, etc.

Muita ênfase, também, tem sido dada ao processo de aprendizagem e compreensão como item a ser mensurado na seleção das características e facilidades de uma linguagem.

Este tipo de experimento pode ser qualificado como uma ferramenta de avaliação de linguagens para efeito de seleção das qualidades desejáveis, proporcionando uma estimativa quantitativa e qualitativa das facilidades de uso. Durante o processo de compra de um novo software para consulta à informações contidas no banco de dados o usuário pode, por exemplo, montar um questionário no qual impõe condições e critérios que devem ser alcançados para que as características da linguagem de consulta se adaptem às suas necessidades operacionais. Neste ponto os testes de aspectos humanos podem ser de grande utilidade.

## 5. ANÁLISE DE LINGUAGENS PARA MANIPULAÇÃO DE BANCOS DE DADOS (PRINCIPAIS S.G.B.D.)

### 5.1 Considerações preliminares

A presente análise visa realizar um levantamento sistemático das características das linguagens de manipulação de banco de dados mais expressiva em cada abordagem de estruturação dos dados, a fim de que se possa obter, desta análise, subsídios suficientes para possibilitar uma comparação e classificação dos sistemas de gerência de banco de dados quando utilizados em uma modalidade de embutimento em linguagens hospedeiras.

Para esta análise foram consideradas as linguagens de manipulação de banco de dados dos seguintes sistemas:

- . TOTAL
- . ADABAS
- . IMS
- . IDMS
- . DMS II
- . INGRES
- . SISTEMA R
- . PASCAL/R.

No decorrer da descrição dos diversos sistemas será usado, para exemplificar a definição da base de dados e também a manipulação dos dados, um banco de dados para a seguinte aplicação, ilustrada numa modelagem entidade-relacionamento (E-R) na figura 5.1.

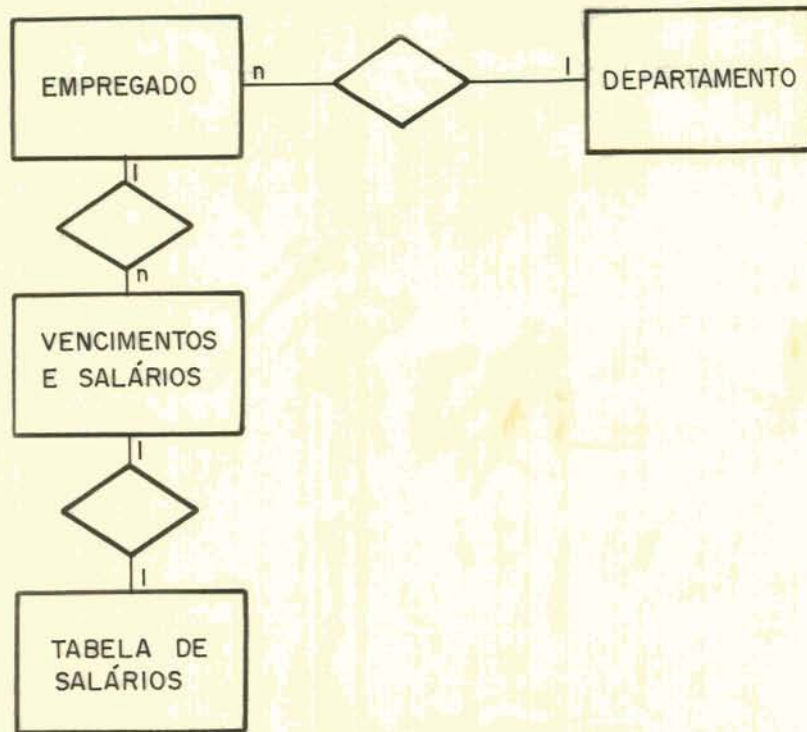


Figura 5.1 - Banco de Dados exemplo

As entidades possuem, como principais atributos, os seguintes:

EMPREGADO (EMP-NRO, EMP-NOME, EMP-DATAADM, EMP-IDADE, EMP-ENDER, EMP-GER, EMP-FUNÇÃO, EMP-LOTAÇÃO)

DEPARTAMENTO (DEP-NRO, DEP-NOME, DEP-LOCAL)

VENC EMP (VENC-ANO, VENC-MES, VENC-EMP, VENC-TAB)

TABELA VENC (TAB-NRO, TAB-NOME, TAB-VALATU)

Na representação destas entidades, relacionamentos e atributos em cada um dos sistemas, certamente adaptações são necessárias pois estarão na dependência das peculiaridades de cada sistema de gerência de banco de dados.

Para que uma descrição dos diversos sistemas de gerência de banco de dados fosse realizada visando uma pos-

terior comparação e classificação das formas de embutimento das linguagens de manipulação de banco de dados, as peculiaridades de cada um dos sistemas foram enquadradas nos seguintes itens de estudo que, direta ou indiretamente, delimitam o ambiente operacional dos sistemas analisados.

a) Quanto à caracterização do sistema

. Estruturas de dados: sob este item procurou-se identificar as estruturas de dados disponíveis no sistema para modelar a realidade.

. Linguagens hospedeiras: neste item foram relacionadas todas as linguagens de programação convencionais, aqui chamadas de linguagens hospedeiras, que alojam a linguagem de manipulação do S.G.B.D. em estudo.

. Estrutura funcional: para que fosse visualizado todo o ambiente operacional do sistema, foi apresentado, sob este item, um diagrama da arquitetura funcional do sistema em questão onde procurou-se ilustrar as diversas interfaces que conectam a comunidade de usuários ao S.G-B.D.

b) Quanto à interface de definição dos dados no sistema

. Definição da base de dados: neste item figuram os aspectos vinculados à definição da base de dados, como por exemplo, a linguagem utilizada para realizar esta tarefa, a complexidade nas especificações e as informações necessárias e relevantes para esta definição (itens, relacionamentos, parâmetros físicos, métodos de acesso).

. Definição de esquemas externos: diz respeito ao subconjunto de informações tornadas disponíveis aos programas de aplicação. São as visões locais da base de dados especificadas para uma determinada aplicação, conhecidas no modelo Codasyl como "Subschema" e em algumas bibliografias

[/ALL 76/, /CHA 75/, /STO 75/, /STO 76/], como vistas ("views"). A terminologia aqui adotada originou-se das convenções propostas pela ANSI/X3/SPARC [/LEI 80/].

c) Quanto aos aspectos funcionais do embutimento

. Comunicação usuário com o banco de dados: neste item é analisado o ambiente do programa de aplicação no que se refere às construções e operadores utilizados para efetivar o embutimento de uma linguagem de manipulação de banco de dados. A comunicação usuário com o banco de dados, quando do embutimento da DML em linguagens hospedeiras, é analisada sob os aspectos de definição dos dados no programa, uso das instruções DML e manipulação das mensagens retornadas ao programa.

. Conversões: para alguns S.G.B.D. existe a necessidade de realizar conversões a fim de adequar o formato dos dados às construções da linguagem hospedeira. Para outros, no entanto, existe uma equivalência de formatos não necessitando nenhum procedimento de conversão. O assunto é discutido neste item para cada sistema.

. Estruturas de controle de fluxo: para que registros do banco de dados sejam transferidos serialmente ao programa de aplicação, existe a necessidade de realizar um controle de fluxo lógico que permita a iteração para recuperação de todos os registros que satisfaçam uma determinada condição, um de cada vez. Para alguns S.G.B.D., este controle é realizado pelo próprio comando DML. Em outros, este controle deve ser realizado pelos comandos da linguagem hospedeira.

. Considerações sobre proteção dos dados: sob este item são discutidas as facilidades que o sistema oferece para garantir a segurança, integridade e proteção dos dados. Para alguns S.G.B.D. estas facilidades estão implícitas no sistema. Em outros elas devem ser especificadas du-

rante a definição dos dados e existe ainda aqueles S.G.B.D. em que as facilidades são explicitamente especificadas por comandos DML.

. Considerações sobre homogeneidade: A homogeneidade diz respeito a uma uniformidade sintática entre as construções da DML e as da linguagem hospedeira e pode refletir uma satisfação pessoal no uso destas construções por parte do programador.

d) Quanto aos aspectos de implementação do embutimento

. Descrição da forma de embutimento: neste item é analisado, para cada S.G.B.D., o mecanismo pelo qual é implementado o embutimento da DML na linguagem de programação convencional.

. Considerações sobre a portabilidade da linguagem: dependendo da forma como o embutimento foi implementado existe uma maior ou menor flexibilidade para permitir que uma DML seja embutida em outras linguagens hospedeiras. Sob este aspecto pode ser determinada a portabilidade da linguagem.

Em vista do PASCAL/R não poder ser enquadrado como um S.G.B.D. que possui a sua DML embutida em uma linguagem de programação convencional, pois todo o gerenciamento do banco de dados é realizado pelo próprio PASCAL/R podendo, assim, ser considerado como uma linguagem autocontida, será efetuada uma descrição não seguindo os itens acima referenciados mas sim identificando as principais características desta linguagem a fim de que possam ser usadas como comparação quando em confronto com os outros S.G.B.D.



## 5.2 O Sistema TOTAL

### 5.2.1 Caracterização do sistema

#### 5.2.1.1 Introdução

O TOTAL [/COH 78/, /DAV 77/, /FUR 82/ e /TOT 82/] é um S.G.B.D. produzido pela CINCOM Systems Inc., estando disponível atualmente, para vários computadores: IBM 360/370, 513-10, S/3-15, ICL 1900 e 2903, Honeywell 200 e 2000, CDC 6000, Univac 70 e 9000, NCR 100, 200 e 300, Cyber 70, PDP 11, etc.

#### 5.2.1.2 Estruturas de dados

O sistema TOTAL utiliza a abordagem em rede (Network) para a estruturação do banco de dados semelhante à proposta CODASYL/DBTG [/TAY 76/] com algumas particularidades no sentido de que um conjunto de dados do tipo mestre em um determinado relacionamento, só poderá ser mestre nos demais relacionamentos e jamais pode ser do tipo membro. O mesmo conceito é aplicado para os conjuntos de dados do tipo membro.

Existem, portanto, sempre 2 níveis na hierarquia.

Os relacionamentos são todos do tipo 1:n, relacionando conjuntos de dados mestre e membro, respectivamente. Os relacionamentos não possuem atributos associados.

Os registros de conjunto de dados mestres são armazenados randomicamente e acessados através de uma chave primária. Os registros de conjunto de dados membros, por sua vez, estão encadeados entre si com apontadores para o anterior e para o sucessor armazenados nos próprios registros.

Um registro mestre aponta para o início e fim da

cadeia de registros membros, através dos campos de apontadores (LINKPATH). Os registros de um conjunto de dados membros podem participar de várias cadeias.

#### 5.2.1.3 Linguagens hospedeiras

A interação do usuário com um banco de dados TOTAL é feita por meio de programas escritos em uma das suas linguagens hospedeiras como COBOL, PL/1, FORTRAN e ASSEMBLER, que referenciam o sistema TOTAL através de chamadas a uma rotina padrão, que faz a comunicação entre o programa e o TOTAL.

Teoricamente o banco de dados TOTAL pode ser acessado por qualquer linguagem hospedeira que tiver implementado o comando "CALL".

A linguagem alvo para a presente descrição será o COBOL.

#### 5.2.1.4 Estrutura funcional

A estrutura funcional do sistema TOTAL pode ser generalizada através da figura 5.2.

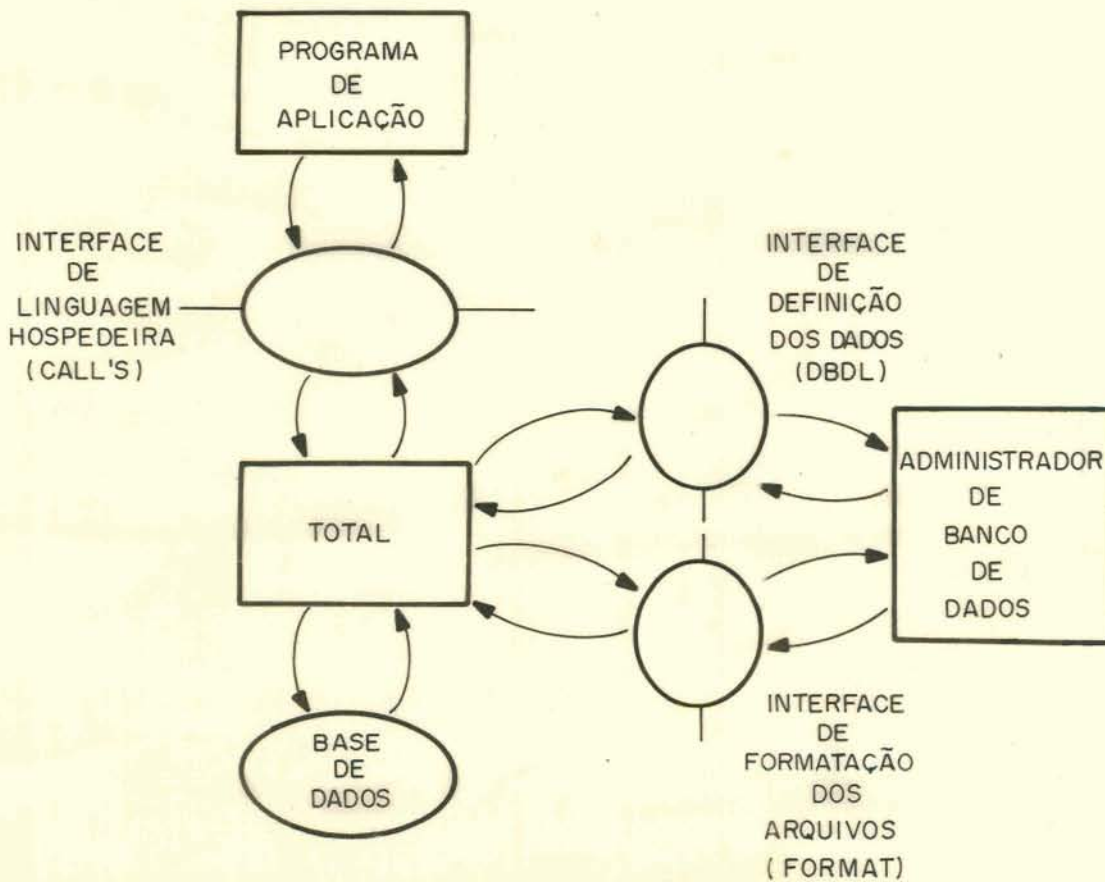


Figura 5.2 - Arquitetura funcional do TOTAL

Detalhes maiores serão analisados no decorrer da descrição funcional e operacional do sistema.

### 5.2.2 Interface de definição dos dados no sistema

#### 5.2.2.1 Definição da base de dados

No sistema TOTAL, a tarefa de definição dos dados fica sob a responsabilidade do administrador do banco de dados.

A descrição do banco de dados é realizada por uma linguagem independente (autocontida) própria do sistema, a DBDL (data base definition language), composta de comandos do tipo palavras-chave.

A definição do banco de dados é feita separadamente dos programas de aplicação que irão acessá-lo. A definição do banco de dados é especificada em termos de conjunto de dados (data set), registro de dados (data record), elementos de dados (data element), relacionamento entre conjunto de dados e também em termos de itens de dados (data item) ou campos.

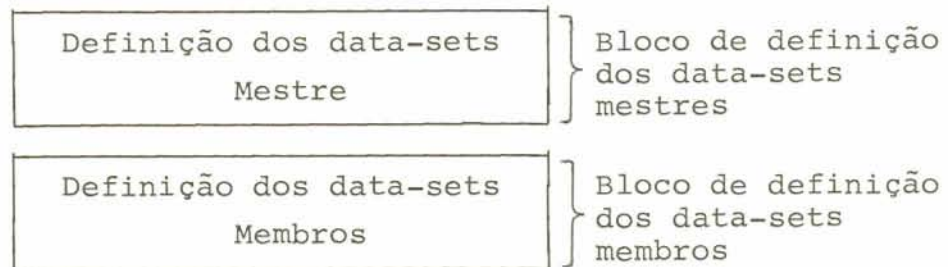
O arquivo fonte, contendo os comandos da DBDL, é usado como entrada para o programa de geração do esquema de definição do banco de dados, conhecido pelo nome DBGEN, o qual gera, a partir das especificações, um texto fonte na linguagem assembler que é montado e armazenado em uma biblioteca, e contém basicamente uma tabela de descrição do banco de dados (DBMOD).

A definição do banco de dados através da DBDL inicia através das especificações gerais sobre este banco de dados onde são definidos o nome do B.D., as áreas de E/S que serão compartilhadas por grupos de data-sets mestres ou grupos de data-sets membros, a proteção para alterações concorrentes e modalidade de utilização como pode ser visto no exemplo a seguir:

```

BEGIN-DATA-BASE-GENERATION:
DATA-BASE-NAME = DBPESS
SHARE-IO:
IOAREA = MEST1
IOAREA = MEST2
IOAREA = MEMB
VERSION = BATCH

```



```

END-DATA-BASE-GENERATION:

```

A definição dos conjuntos mestres e membros consiste na especificação do nome e da área de entrada/saída, na definição dos elementos que compõem o registro ou data-set e na definição do ambiente físico relativo ao conjunto como dispositivo de armazenamento, número de registros lógicos, tamanho do registro, fator de bloco, etc.

Para visualizar uma definição completa de conjuntos de dados segue um exemplo:

CONJUNTO MESTRE

```

BEGIN-MASTER-DATA-SET:
DATA-SET-NAME = DEPT      - Nome c/4 caracteres
IOAREA = MEST2           - Área de E/S c/4 caracteres
MASTER-DATA:            * Início da descrição do registro
DEPTROOT = 8             - Campo p/manipul.de sinônimo
DEPTCTRL = 6            - Chave
DEPTLKLT = 8            - Ligação p/arquivo de lotação
DEPTDADOS = 60
  2 DEPTNOME = 30
  2 DEPTLOCL = 30
END-DATA:                * Término da descr. do registro
DEVICE = 2314            - Definição das caracter. físicas
TOTAL-LOGICAL-RECORDS = 10000
LOGICAL-RECORD-LENGHT = 82
LOGICAL-RECORDS-PER-BLOCK = 100
LOGICAL-BLOCKS-PER-TRACK = 1000
TOTAL-TRACKS = 10
END-MASTER-DATA-SET:

```

CONJUNTO MEMBRO

```

BEGIN-VARIABLE-ENTRY-DATA-SET:
DATA-SET-NAME = LOTA
IOAREA = MEMB
BASE-DATA:              * Início da descrição do registro
LOTADEPT = 6 = DEPTCTRL - Chave do arquivo DEPT
LOTAEMPR = 6 = EMPCRTL  - Chave do arquivo EMPR
DEPTLKLT = 8            - Ligação entre membros do DEPT
EMPRLKLT = 8            - Ligação entre membros de EMPR
LOTADADOS = 10
  2 LOTAFUNC = 4
  2 LOTAHORA = 6
END-DATA:                * Término da descr. do registro
IDEM DATA-SET MESTRE
CYLINDER-LOAD-LIMIT = 80
END-VARIABLE-DATA-SET:

```

Na definição dos nomes dos elementos de dados (campos) de um conjunto de dados o nome do data-set entra como prefixo na formação destes nomes que serão compostos de 8 caracteres. Estes elementos de dados podem ser, opcionalmente, precedidos por número de nível com valor 1 a 99.

Para um registro de dados membro devem ser definidos tantos campos CTRL quantos são os conjuntos mestres, dos quais ele é membro e um par (CTRL,LK) é definido para cada ligação a qual o membro pertença.

Um conjunto de dados mestre pode ter apenas um formato. Já um conjunto membro pode assumir mais de um formato, cada um deles identificado por um código (redefinição de registros). Os diferentes tipos de registros são especificados pela cláusula RECORD CODE.

#### 5.2.2.2 Definição de esquemas externos

No sistema TOTAL não existe uma definição de esquema externo propriamente dito. A implementação de esquemas externos é realizada a nível de programa de aplicação por um parâmetro da linguagem de manipulação de dados que contém os nomes dos elementos de dados a serem acessados pelo programa.

A especificação pode ser feita como mostra o exemplo abaixo, escrito na linguagem COBOL.

```
WORKING-STORAGE SECTION
:
01 LISTA-ELEMENTOS.
02 ELEM-DEPT PIC X(23) VALUE
    "DEPTCTRLDEPTNOMEDEPTLOCLEND."
```

O método de definição de esquemas externos a uma

aplicação não oferece muita independência entre o programa de aplicação e os arquivos físicos, uma vez que a definição de um registro lógico só pode se basear em um registro físico [LEI 80/].

### 5.2.3 Aspectos funcionais do embutimento

#### 5.2.3.1 Comunicação do usuário com o banco de dados

Para o sistema TOTAL, a área de comunicação do usuário com o sistema de gerência de banco de dados é constituída dos elementos que aparecem na figura 5.3.



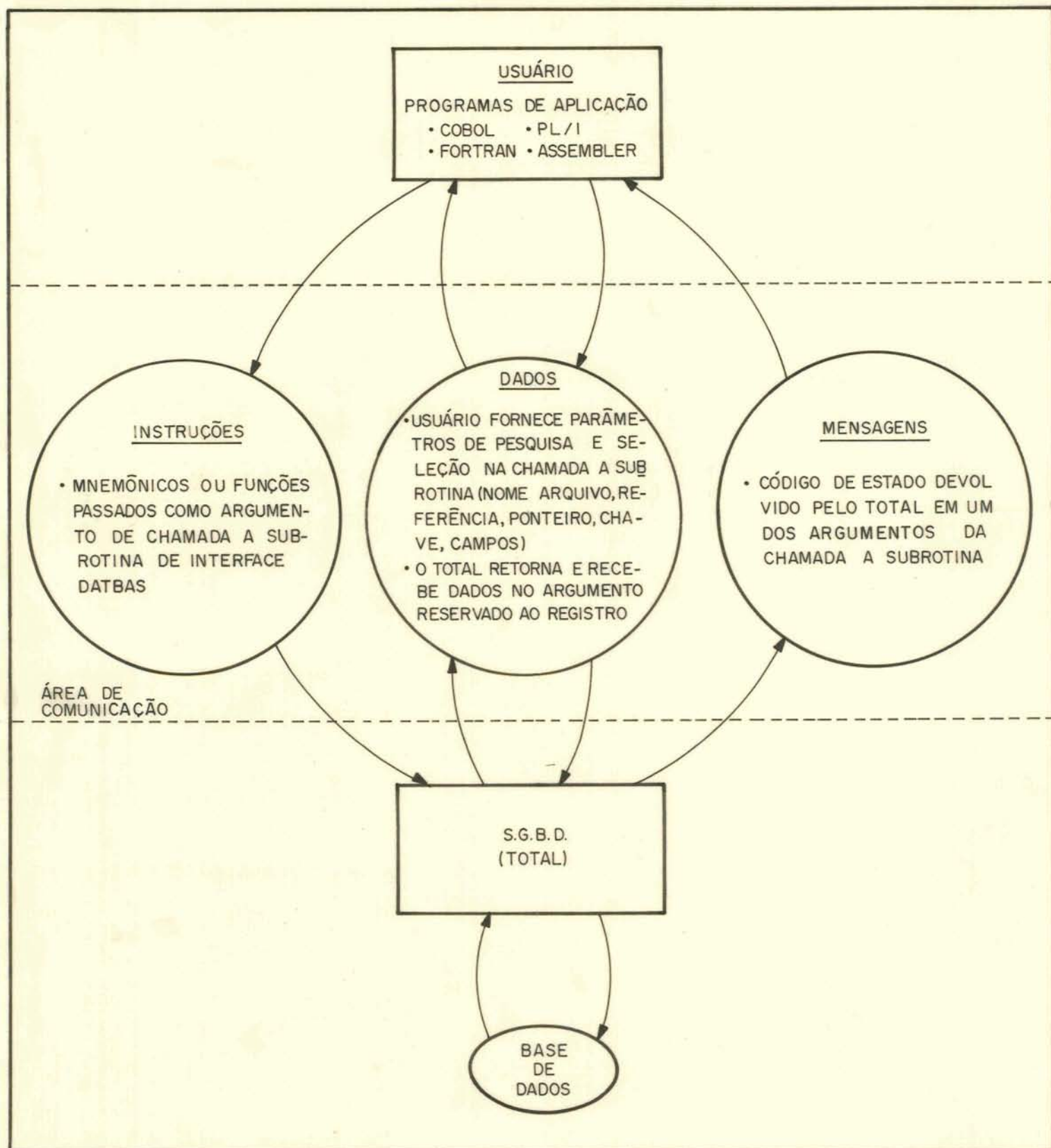


Figura 5.3 - Canais de comunicação programas de aplicação e TOTAL

Toda a comunicação entre o sistema e o usuário é realizada através da macro "CALL" de chamada da rotina padrão de interface DATBAS.

Os argumentos passados como parâmetros desta chamada à subrotina conterão as informações necessárias para fornecimento da operação a ser executada; recepção de código de estado; fornecimento dos atributos de pesquisa e seleção e transmissão e recepção dos dados compartilhados entre o programa de aplicação e o sistema TOTAL.

A utilização da interface com o sistema TOTAL é dividida em 2 etapas distintas:

. Definição de constantes, variáveis, registros lógicos e áreas de entrada e saída na seção de definição de dados do programa (Data Division COBOL ou DCLs PL/1);

. Codificação dos CALLs correspondentes às funções que se deseja executar.

a) DADOS

O fluxo de dados entre o programa de aplicação e o sistema TOTAL é estabelecido através da interface DATBAS, materializado no programa de aplicação com a chamada a esta subrotina pelo comando CALL.

Para o estabelecimento deste fluxo de dados o usuário fornece parâmetros de pesquisa e seleção no acesso a banco de dados como nome do data-set, referência (posição relativa do registro), ponteiros, chave, nome dos campos envolvidos na operação, além de estabelecer, também, uma área na qual devem ser colocados ou da qual devem ser retirados os elementos solicitados na operação.

O usuário é responsável pela definição do layout do registro no programa de aplicação para as operações de entrada/saída sobre o banco de dados.

Para especificar quais os campos que o programa de aplicação manipulará, o usuário deverá definir, no parâmetro adequado (parâmetro "elementos") a seqüência de nomes de campos, cada qual com 8 caracteres e todos fazendo parte de um mesmo data-set, que deverão ser considerados na operação. Os dados, relativos aos campos especificados, estarão contidos num outro parâmetro num layout e formato previamente definido pelo administrador do banco de dados através da DBDL e cuja máscara (imagem) deste formato deverá estar, também, descrita no programa de aplicação, sob responsabilidade do usuário.

O exemplo que segue, como usado na linguagem COBOL, ilustra a descrição dos dados no programa de aplicação.

WORKING-STORAGE SECTION

```

      :
01  COMANDOS.
      02  READM PIC X(5) VALUE "READM".

01  LISTA-ELEMENTOS.
      02  ELEM-DEPT PIC X(23) VALUE "DEPTCTRL DEPTNOME
                                DEPTLOCL END.".
01  ARQUIVOS.
      02  ARQ-DEPT PIC X(4) VALUE "DEPT".

01  AREAS-ES.
      02  LAYOUT-DEPT.
          03  DEP-NRO PIC X(6)
          03  DEP-NOME PIC X(30)
          03  DEP-LOCAL PIC X(30)

01  OUTRAS-VARIAVEIS.
      2  STATUS PIC X(4).
      2  FIM-PAR PIC X(4) VALUE "END.".
      :
CALL  DATABAS USING READM, STATUS, ARQ-DEPT, CHAVE,
                                ELEM-DEPT, LAYOUT-DEPT, FIM-PAR.
      :

```

## b) INSTRUÇÕES

As instruções são passadas ao sistema de gerência de banco de dados TOTAL através de mnemônicos indicativos do código de operação ou função colocados no primeiro parâmetro da chamada à subrotina de interface e definirão as funções que serão executadas através desta macro "CALL".

Exemplo:

```
CALL DATBAS (<parâmetro 1>, <parâmetro 2>, <parâmetro 3>, ..., <parâmetro 9>)
```

O <parâmetro 1> é o código de operação de 5 caracteres como, por exemplo, "READM" que indicará uma leitura de um conjunto mestre.

As operações sobre o banco de dados podem ser do tipo:

- Abertura e fechamento de conjuntos de dados (OPENM, CLOSEM, ...);
- Acesso serial (SEQRM, SEQRV, ...);
- Acesso direto a mestres (READM, WRITM, ADD-M, ...);
- Acesso direto a membros (READR, READV, ...);
- Controle de alocação (TOTAL, MPTOT, QUEST, ...).

As instruções funcionam a nível de elementos de dados.

Aliado ao parâmetro, que informa o código de operações, outros argumentos passados como parâmetro indicarão o critério de seleção de registros.

A recuperação dos dados pode ser feita serialmente (ordenação lógica) ou diretamente. No caso dos conjuntos mestres o acesso direto se realiza através de cálculo do endereço (HASH) ao contrário do caso dos conjuntos membros onde este acesso é realizado por meio dos apontadores.

A forma geral de uso da macro "CALL" é:

Em PL/1:

```
CALL DATBAS (<operação>,<estado>,<arquivo>,  
             <referência>,<ligação>,<chave>,  
             <elementos>,<registro>,<fim>);
```

Em COBOL:

```
CALL 'DATBAS' USING <operação>,<estado>, ...
```

Cada parâmetro representa um endereço que será passado ao TOTAL e devem ser codificados numa ordem física preestabelecida se bem que, nem todos os parâmetros são necessários, podendo a lista ser interrompida em lugares permissíveis por meio do uso do parâmetro que indicará fim da lista ('END'). O número de parâmetros exigidos dependerá da operação que está sendo desejada.

Os argumentos ou parâmetros passados a interface DATBAS possuem a seguinte finalidade:

- . <operação> É o nome da função, em forma de mnemônico, de 5 caracteres que define a instrução a ser executada.
- . <estado> É uma variável que conterà um indicador, em forma de mnemônico de 4 caracteres, sobre o resultado da operação.
- . <arquivo> Especifica qual a entidade (conjunto de dados) será utilizada. Deverá ser uma variável que comporte 4 caracteres.
- . <referência> Área, de tamanho equivalente a 4 bytes, onde o TOTAL colocará o endereço físico do registro que está sendo processado, ou de seu antecessor, quando for uma operação de elimi-

nação, ou ainda a constante 'END' em caso de fim de processamento da cadeia.

- . <ligação> É o nome de um relacionamento, com 8 caracteres, como declarado em um campo tipo LK da DBDL e indicará o caminho de acesso a ser seguido.
- . <chave> Corresponde a chave primária do registro mestre e é usado nas operações não seriais tanto sobre os conjuntos mestres como sobre os conjuntos membros.
- . <elementos> Lista de nomes dos elementos, todos com 8 caracteres, que formam o registro lógico a ser utilizado. A seqüência de nomes termina com um 'END'.
- . <registro> É o nome da área do usuário para entrada e saída para comunicação dos dados entre o programa de aplicação e o TOTAL.
- . <fim> campo de 4 caracteres contendo a constante 'END' para indicar o final da lista dos parâmetros.

Para ilustrar os tipos de operações, é apresentado, na figura 5.4, um quadro com as principais funções:

TIPO	MNEMÔNICO	FUNÇÃO
ABERTURA E FECHAMENTO DE CONJUNTOS	OPENM E CLOSM	- ABERTURA E FECHAMENTO DE MESTRE
	OPENV E CLOSV	- ABERTURA E FECHAMENTO DE MEMBRO
	OPENX E CLOSX	- ABERTURA E FECHAMENTO DE LISTA DE CONJUNTOS
ACESSO SERIAL	SEQRM	- RECUP. SERIAL DE MESTRES
	SEQLV	- RECUP. SERIAL DE MEMBROS
	SEQRV	- RECUP. SEQÜENCIAL DE MEMBROS, SEGUINDO A LIGAÇÃO
	SEQWV	- ATUALIZA REGISTRO ACESSADO, POR SEQRV OU SEQLV
	RESTM	- INICIALIZA APONTADOR DE REFER. P/ MESTRES
	RESTV	- INICIALIZA APONTADOR DE REFER. P/ MEMBROS
ACESSO DIRETO A MESTRE	READM E WRITM	- LEITURA E GRAVAÇÃO ATRAVÉS DE CHAVE
	ADD-M E DEL-M	- INSERÇÃO E ELIMINAÇÃO DE REGISTROS ATRAVÉS DE CHAVE
ACESSO DIRETO A MEMBRO	READV E READR	- OBTÉM REGISTRO SEGUNDO A CADEIA DE UM RELACIONAMENTO DO INÍCIO AO FIM OU VICE-VERSA
	READD E WRITV	- RECUPERA OU ARMAZENA O REGISTRO DIRETAMENTE PELO EN DEREÇO EM <REFERÊNCIA>
	ADDVC	- INSERE NOVO REGISTRO NO FIM DA CADEIA
	ADDVB E ADDVA	- ADIÇÃO DE REGISTRO DEPOIS OU ANTES DE UM REGISTRO LIDO NA LIGAÇÃO <CHAVE> <LIGAÇÃO>
	DELVD	- ELIMINA FISICAMENTE O ÚLTIMO REGISTRO
CONTROLE DE ALOCAÇÃO	TOTAL	- INICIALIZAÇÃO / CARGA DO TOTAL
	MPTOT	- CARGA DE APENAS O MÓDULO DE LEITURA E ALTERAÇÃO DE REGISTROS
	QUEST	- CARGA DE APENAS O MÓDULO DE LEITURA DE REGISTROS
	DEQUE	- FORÇA A EFETIVAÇÃO DAS ALTERAÇÕES, FINALIZAÇÃO
FUNÇÕES DE CONTROLE DE SEGURANÇA	LOGMARK	- INSERÇÃO DE REGISTRO NO ARQUIVO LOG
	LOGQUIET	- COMPLETA OPERAÇÕES DE ATUALIZAÇÃO ANORMAIS E REPORTA ANOMALIAS NO LOG
	PURGE	- ELIMINA DA MEMÓRIA OS MÓDULOS NÃO SENDO UTILIZADOS
	RESERVE	- EXCLUSÃO MÚTUA DE ARQUIVO (BLOQUEIO)
	SHARE	- COMPARTILHAMENTO DE UM ARQUIVO POR DIVERSOS USUÁRIOS
	RELEASE	- LIBERA BLOQUEIO

Figura 5.4 : Quadro de instruções DML do TOTAL

Detalhes maiores sobre as operações ou funções da linguagem de manipulação de dados podem ser obtidos na bibliografia [/DAV 77/].

Convém salientar que a obtenção de registros é feita segundo o protocolo "um-de-cada-vez", isto é, não pode haver, através de um único comando, obtenção do acesso a um conjunto de registros. Nos comandos só pode haver referência a uma única entidade. Para obter o acesso a diversas entidades é necessário criar uma lógica no programa com a utilização de diversos comandos de recuperação dos dados.

#### c) MENSAGENS

Junto com os argumentos para instruções e dados na chamada à subrotina DATBAS, o usuário deverá definir uma variável que receba um código de estado devolvido pelo TOTAL após a operação e que informa como esta operação se desenvolveu.

Como exemplo dos códigos retornados pode-se mencionar:

- . '\*\*\*\*' - operação bem sucedida
- . 'DUPM' - registro mestre duplicado
- . 'IPAR' - lista de parâmetros inválidos
- . 'FULL' - arquivo esgotado
- . 'MRNF' - registro mestre não localizado

#### 5.2.3.2 Conversões

O TOTAL não mantém informações sobre o tipo de dado relativo a cada campo do registro. Existe somente uma especificação do tamanho de cada campo em número de bytes, durante a definição dos dados. Todos os campos, portanto, são tratados da mesma forma.



Em vista disto, não existe nenhum mecanismo de conversões implementado no sistema. Fica, portanto, sob inteira responsabilidade do usuário a declaração correta dos campos que utilizará no programa de aplicação.

Na formação de registros lógicos, o total só permite que estes sejam formados a partir de elementos (campos) de um único registro físico, ou seja, um registro lógico é formado pela projeção (total ou parcial, com mesma ordem ou ordem invertida) dos elementos de um único registro físico, e nenhuma verificação de formato é feita sobre este registro lógico.

Verifica-se, com isto, que o usuário deverá ter total conhecimento do layout de um registro em um conjunto de dados.

#### 5.2.3.3 Estruturas de controle de fluxo

Em vista do sistema TOTAL realizar a recuperação dos registros num protocolo "um-de-cada-vez" cabe ao usuário codificar em seu programa um fluxo de procedimentos iterativos para a obtenção de todos os registros desejados. Deverá, também, tomar o cuidado para que, a cada nova iteração (loop com uma chamada à subrotina de interface), seja testado o código de estado da operação a fim de controlar, além de outras exceções, a condição de fim de arquivo.

#### 5.2.3.4 Considerações sobre a proteção dos dados

Nos aspectos de segurança o sistema TOTAL deixa muito a desejar, visto que nenhum mecanismo de autorizações ou restrições seletivas para o acesso de um usuário às informações contidas no banco de dados, está implementado. Nenhum controle de acesso aos dados é oferecido pelo sistema.

Com relação ao aspecto de definição de esquemas externos como um mecanismo para garantir certa segurança ao sistema, as características estruturais dificultam maiores aspirações neste sentido, visto que, como já mencionado anteriormente, a implementação de visões locais fica ao encargo do próprio usuário que as define no programa de aplicação através da lista de elementos que deseja recuperar de um determinado registro do banco de dados.

No controle de integridade, o sistema também não detém grandes facilidades. Um exemplo disso, seria o fato de que nenhum controle sobre o formato dos dados manipulados é realizado, pois o TOTAL não tem informações sobre os tipos e estados possíveis dos campos [ /DAV 77/ ].

Apesar disso, pode-se considerar alguns aspectos como:

. Informações estruturais que são armazenadas fisicamente junto com os registros, não são tornadas disponíveis ao usuário. O TOTAL só permite aos programas de aplicação acessar os campos dos registros lógicos.

. Alguns controles sobre inclusões, deleções e modificações de registros são realizados. O TOTAL, por exemplo, detecta tentativas de inserir um registro com chave primária em branco, inserir um registro mestre com chave duplicada, acessar arquivos ou elementos da base de dados que não tenham sido definidos e deletar um registro mestre que ainda está relacionado com um ou mais registros membros.

No que tange ao mecanismo de proteção pode-se levar em conta o aspecto de que o programador deve especificar em qual modo o programa irá executar: somente leitura ou atualização em recovery. O TOTAL assegura que somente os comandos concernentes ao modo escolhido, serão executados.

Problemas de atualizações simultâneas podem ser evitados através das funções de controle RESERVE e RELEASE

que garantem a exclusividade ou não no acesso a arquivos. Para recuperação em caso de falhas, o TOTAL oferece a possibilidade de criação de um arquivo contendo as transações (imagem anterior e posterior) feitas sobre o banco de dados e informações de controle, tais como checkpoints do sistema (pela especificação de um intervalo para realização de todas as operações de entrada/saída pendentes) ou checkpoints do usuário, pelo comando LOGMARK.

O TOTAL não fornece aos usuários nenhum mecanismo de recuperação automática em caso de término anormal de programa. A tarefa de reinício é de responsabilidade do usuário.

#### 5.2.3.5 Considerações sobre homogeneidade

Nos programas de aplicação, a forma de manipulação dos dados e conseqüente chamada a interface DATBAS, segue os padrões de sintaxe da linguagem hospedeira em que este programa está escrito e é dependente, portanto, desta linguagem específica. Existe uma uniformidade de tratamento de chamada a interface S.G.B.D. com a da chamada a rotinas externas convencionais e a sintaxe desta chamada vai depender da linguagem hospedeira que está sendo utilizada.

#### 5.2.4 Aspectos de implementação do embutimento

##### 5.2.4.1 Descrição da forma de embutimento

A implementação do embutimento de instruções do TOTAL nas linguagens hospedeiras se efetiva de uma forma muito simplificada, ou seja, tratando este embutimento como se fosse uma chamada a uma subrotina externa convencional passando como parâmetro os argumentos para que a tarefa executada pela subrotina externa esteja de acordo com o espera

do, retornando o resultado adequado.

O programa de aplicação se comunica com a interface do TOTAL, denominada DATBAS, através de macros CALL, sendo que esta interface, em contato com o núcleo principal do TOTAL (primitivas de acesso e tabelas de descrição), interpreta e executa as operações solicitadas, acessando a base de dados para recuperar ou armazenar as informações que são compartilhadas pelos 2 processos através de uma área de dados que o usuário define no programa.

Portanto, em relação às interfaces oferecidas para os diversos tipos de usuários, o TOTAL só oferece a interface para acesso por uma linguagem hospedeira como COBOL, PL/1, FORTRAN ou ASSEMBLER, implementada, como já foi visto, pela utilização da macro CALL para chamada da rotina DATBAS.

#### 5.2.4.2 Considerações sobre a portabilidade da linguagem

Tendo em vista a simplicidade da forma de embutimento e do uso padronizado do formato dos dados manipulados pelo sistema, evitando a necessidade de conversões explícitas ou implícitas destes formatos, a tarefa de adaptar o uso do sistema em outras linguagens hospedeiras se restringe somente ao fato desta nova linguagem hospedeira já possuir ou não a facilidade de chamadas a rotinas externas ao programa (algumas linguagens podem não serem concebidas para esta finalidade).

### 5.3 O Sistema ADABAS

#### 5.3.1 Caracterização do sistema

##### 5.3.1.1 Introdução

O ADABAS (Adaptable Data Base System) é um sistema de gerência de banco de dados [/DAV 77/, /FUR 82/ e /HEU 82/] produzido pela firma alemã SOFTWARE AG, sendo oferecido para os computadores IBM 360/370, SIEMENS 4004 e UNIVAC-70.

##### 5.3.1.2 Estruturas de dados

O ADABAS utiliza arquivos invertidos para acesso e relacionamento entre arquivos não existindo uma estrutura lógica preestabelecida como nos sistemas que seguem a abordagem em rede/hierárquica. Trata-se de uma coleção de arquivos, fazendo o papel de entidade e onde os relacionamentos são montados com base em atributos iguais para as entidades relacionadas.

Um acoplamento lógico entre arquivos permite a definição de bancos de dados estruturados nos moldes da abordagem hierárquica e em redes [/FUR 82/].

No ADABAS, arquivos, registros e campos mantêm seu significado tradicional e cabe ao usuário a tarefa de estruturação, devendo para tanto especificar todos os relacionamentos lógicos desejados.

A utilização de uma arquitetura baseada em listas invertidas caracteriza o ADABAS como um sistema de gerência de banco de dados voltado para aplicações que utilizem dados pouco voláteis sobre os quais exista uma grande necessidade de recuperações orientadas.

Como facilidade do sistema pode ser mencionado o fato do sistema ADABAS possuir a capacidade de definir uma larga variedade de tipos de dados e estruturas no banco de dados e possuir, também, a capacidade de estabelecer novos campos em arquivos já existente e a capacidade de criar um novo arquivo com partes constituintes do sistema sem a necessidade de uma recarga para restabelecer o relacionamento lógico (file coupling).

O fato de não existir um modelo pré-definido oferece uma certa flexibilidade no projeto das estruturas de dados, mas implica numa necessidade de especificação explícita de todos os relacionamentos e acarreta numa sobrecarga na utilização de espaço de armazenamento, uma vez que os relacionamentos entre arquivos são mantidos pelas listas invertidas e exigem que exista um campo comum aos dois arquivos relacionados (redundância de informações).

#### 5.3.1.3 Linguagens hospedeiras

O ADABAS está disponível para o uso em linguagens hospedeiras como COBOL, PL/1, FORTRAN e ASSEMBLER através de sua interface para o acesso ao banco de dados cuja implementação é realizada pela utilização da macro CALL codificadas nos programas de aplicação.

Além de uma interface com as linguagens hospedeiras, o ADABAS provê, também, recursos independentes destas, como o de uma linguagem interativa, denominada ADASCRIP T +, que permite a interação de usuários casuais com o banco de dados. Adicionalmente oferece a facilidade de emissão de relatórios pelo ADAWRITER. Outra linguagem destinada a usuários casuais é a ADACOM, que também tem características de uma linguagem interativa. Esta se baseia na linguagem BASIC e incorpora o ADASCRIP T e o ADAWRITER [/DAV 77/].

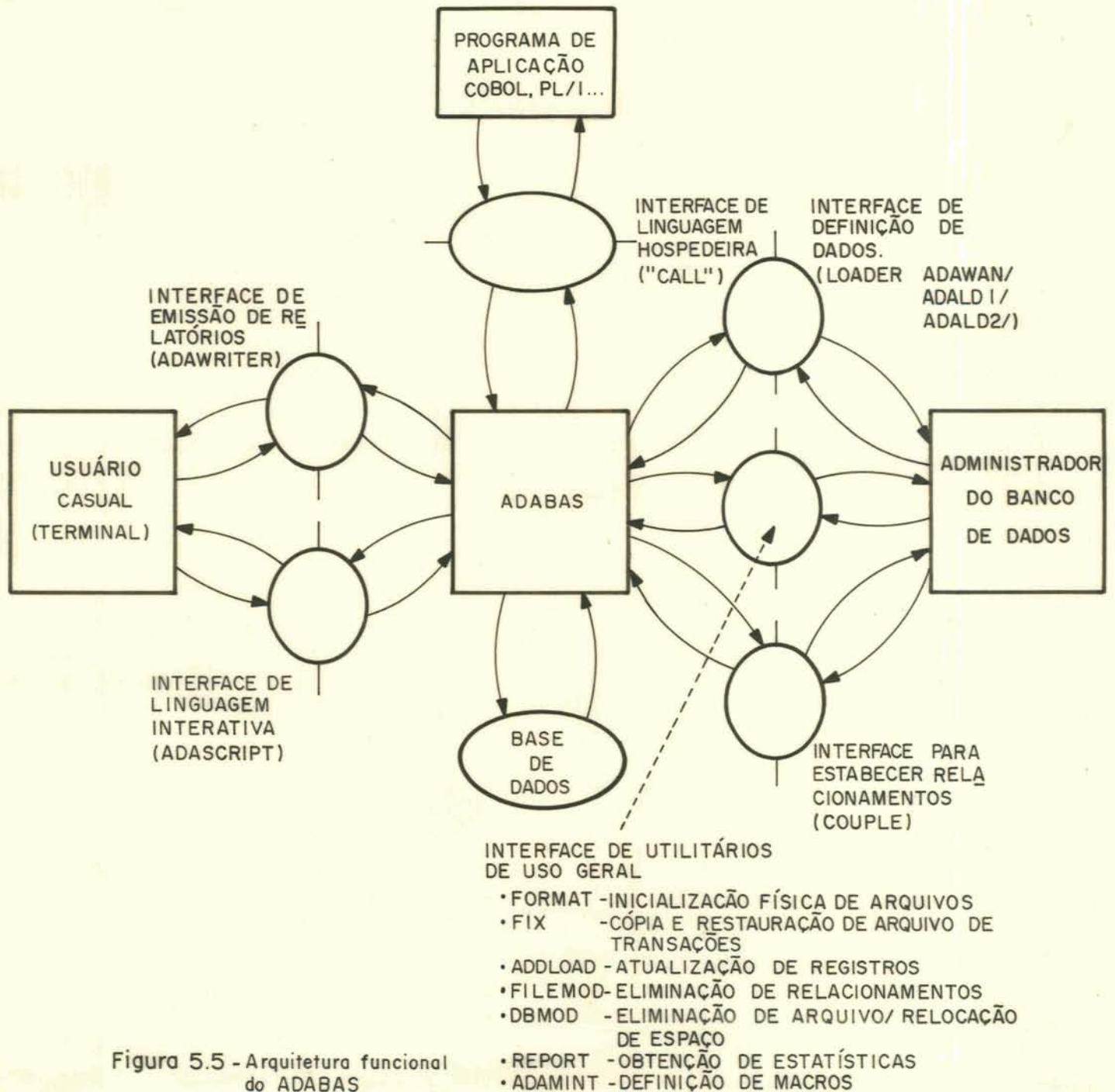
Recentemente foi desenvolvida uma linguagem de

programação de propósitos gerais denominada NATURAL [ /SOF 82/ ] que sustenta uma interface com o sistema ADABAS através de comandos específicos de alto nível (FIND, READ, HISTOGRAM, UPDATE, DELETE, BEGIN OF TRANSACTION, etc.) que fazem parte do acervo de comandos dessa nova linguagem. Para esta linguagem poderia ser estabelecida uma analogia com a extensão da linguagem COBOL para interface com o sistema DBTG da CODASYL [ /TAY 76/ ], por exemplo, se bem que a linguagem NATURAL já incluiu, em seu projeto, as especificações de interface com o ADABAS. A linguagem NATURAL é uma linguagem completa de programação com recursos de manter uma interface autocontida com o sistema ADABAS.

A linguagem alvo para esta descrição será o COBOL com o embutimento da DML baseado em chamadas à subrotina através da macro CALL.

#### 5.3.1.4 Estrutura funcional

Abstraindo maiores detalhes de cada módulo, que serão discutidos no decorrer da descrição do sistema, é apresentado, na figura 5.5, um diagrama sintético da estrutura funcional do ADABAS.



A área de interesse para esta análise será somente a das interfaces de linguagem hospedeira, de definição de dados e relacionamentos e de definição de esquemas externos (visões locais).



### 5.3.2 Interface de definição dos dados no sistema

#### 5.3.2.1 Definição da base de dados

O sistema possui uma linguagem de definição de dados própria, cujos comandos são divididos em 2 grupos - de definição de arquivos e de definição de relacionamentos (acoplamentos) - e oferece dois programas para a geração das tabelas de descrição do banco de dados (loader & couple) a partir das especificações feitas pelos comandos da linguagem de definição. As tabelas são usadas pelo ADABAS para atender aos comandos especificados pelos programas de aplicação.

A definição do banco de dados ADABAS consiste na declaração de arquivos, campos de seus registros e relacionamentos entre arquivos.

Qualquer campo, em um arquivo ADABAS, pode ser designado como chave primária (descriptor) na criação ou durante a utilização do arquivo [/FUR 82/].

A linguagem de definição dos dados (DDL) consiste de uma série de comandos que descrevem os campos do arquivo que é definido por ocasião de sua carga, recebendo um número que o identifica univocamente dentro do banco de dados. O programador deverá ter conhecimento deste número quando de-sejar utilizar a referida entidade em seu programa.

Campos dos registros podem ser definidos na carga do arquivo ou adicionados dinamicamente aos registros após esta carga sem reflexos sobre os programas já referidos.

Relacionamentos hierárquicos entre campos de um registro podem ser estabelecidos através da definição de grupos de campos que torna possível a referência a um agregado de dados de forma compacta.

A definição de arquivos em ADABAS, portanto, en-

volve especificação das entradas de descrição de campo para cada campo no arquivo, obedecendo a seguinte sintaxe:

<nível>,<nome>[,<tamanho>][,<formato>][,<opções>]

onde:

- . <nível>           Especifica o nível do campo dentro de uma estrutura hierárquica (1 a 7).
- . <nome>           É o identificador do campo, formado por 2 caracteres e deve ser único no arquivo.
- . <tamanho>        Tamanho padrão do campo que será usado quando um programa de aplicação nada indicar. O tamanho é indicado em número de bytes.
- . <formato>        Formato dos campos de nível elementar a ser usado como formato de armazenamento e carga inicial via utilitário. Os formatos são os seguintes:
  - . A - Alfanumérico
  - . B - Binário
  - . F - Ponto fixo
  - . G - Ponto flutuante
  - . P - Decimal compactado
  - . U - Decimal não compactado
- . <opções>        Especificação de condições especiais do campo. As opções são as seguintes:
  - . DE - Descritor (possuirá lista invertida)
  - . FI - Tamanho fixo (não pode ser compactado)
  - . NU - Supressão de valores nulos (zeros,brancos)
  - . MU - Campo com número variável de valores num registro
  - . PE - Grupos periódicos ("occurs"), permiti dos somente em nível 01.

A descrição dos campos é processada pelo utilitário LOADER que se encarrega de realizar:

- . Análise da definição lógica do arquivo, edição e compressão dos dados (rotina ADAWAN).

- . Alocação e formatação de arquivos físicos, definição de características físicas e carga da tabela de definição e dos dados (rotina ADALD1).

- . Carga das listas invertidas (rotina ADALD2).

Relacionamento explícito pode ser estabelecido através do acoplamento de arquivo (file coupling). Um campo descritor comum pode ser usado para relacionar 2 arquivos tal que os atributos de ambos os arquivos possam ser comparados nos comandos de pesquisa. Este relacionamento é processado pelo utilitário COUPLE e a sintaxe para realizar o intento é a seguinte:

```
FILE <nro.arq>(<nome campo>) WITH FILE
      <nro.arq>(<nome campo>)
```

A cada registro de um arquivo é associado um número de seqüência interno (ISN - Internal Sequence Number) que, uma vez atribuído, nunca muda, e é usado para prover o acesso aos registros e para fazer o relacionamento entre os arquivos.

Para visualizar melhor a descrição de dados e relacionamentos, é apresentado um exemplo:

01,DP	* Descrição do arquivo departamento
02,DN,6,B,DE	Nro. do departamento
02,DD,30,A,NU	Denominação
02,DL,30,A,NU	Local

01,EP	* Descrição do arquivo empregado
02,EN,6,B,DE	Nro. do empregado
02,E1,30,A,NU	Nome
02,ED,6,B,DE	Nro. do departamento
02,ES,11,G,FI	Salário
02,EG,6,B,FI	Gerente
02,EI,2,B,NU	Idade

FILE 01(DN) WITH FILE 02(ED)      \* Descrição do relacionamento

#### 5.3.2.2 Definição de esquemas externos

Não existe, no ADABAS, propriamente, uma definição de esquemas externos, visto que este conceito, como proposto pela ANSI/X3/SPARC [/LEI 80/], não pode ser literalmente aplicado para este sistema. Durante a definição da base de dados, nenhuma descrição de subconjuntos de informações, limitadas a determinada aplicação, é especificada para que possa ser, posteriormente, referenciada pelos programas de aplicação.

O que existe no ADABAS, e que alguns autores consideram como vistas ("views") [/FUR 82/], é a especificação de macros (ADAMINT - ADABAS MACRO INTERFACE) que permite ao administrador do banco de dados gerar restrições de acesso para um usuário (visão local). A definição é feita na geração do módulo de acesso com base em parâmetros que delineiam esta definição. Este módulo de acesso conterà um ponto de entrada para cada função definida. Os nomes dos pontos de entrada são referenciados nos programas de aplicação, para invocar as funções correspondentes e é através do conjunto de parâmetros predeterminados da macro CALL que o usuário poderá usar a visão local gerada.

Visões locais podem também ser definidas pelo prô

prio programador, porém sem grandes restrições de auxílio à segurança. Pode ser feita dentro do programa de aplicação e consiste em um conjunto de especificações de registros lógicos definidos pelo programador na área de formato (format buffer) composto por especificações de campos pelo seu nome, tamanho em bytes e tipo de representação. O programador pode especificar quantos campos desejar, sendo também permitido o uso de espaços entre os campos.

Para exemplificar esta forma de uso de visões um registro, da entidade Empregados, formado pelos campos número, nome, salário e idade, pode ser especificado na seguinte forma:

```
⋮  
MOVE "EN,4X,E1,20,4X,ES,4X,EI" TO AREA-DE-FORMATO  
⋮  
CALL 'ADABAS' USING ...
```

### 5.3.3 Aspectos funcionais do embutimento

#### 5.3.3.1 Comunicação do usuário com o banco de dados

A área de comunicação do usuário com o sistema de gerência de banco de dados aparece esboçada na figura 5.6.

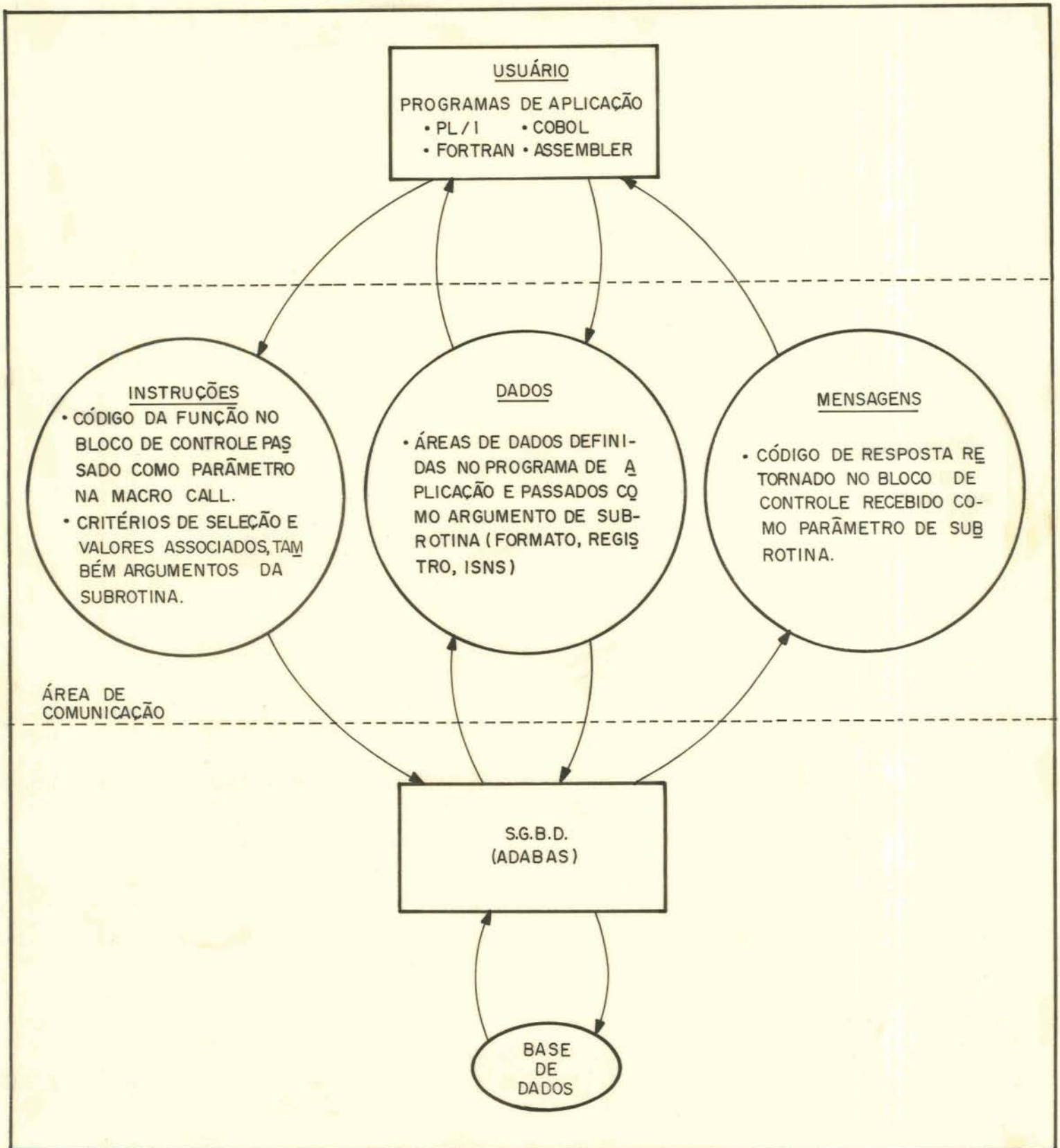


Figura 5.6 - Canais de comunicação programas de aplicação e ADABAS

Os recursos necessários para a comunicação entre os programas de aplicação e o sistema de gerência de banco de dados são proporcionados através da linguagem de manipulação de dados. Esta comunicação é realizada sob a forma de chamadas a uma subrotina denominada 'ADABAS'.

A utilização da interface de linguagem hospedeira 'ADABAS' consiste na declaração e inicialização de seis áreas de comunicação a serem utilizadas como parâmetros na chamada do módulo de interface. Estas seis áreas específicas, definidas para serem os argumentos nas chamadas à subrotina de interface, constituirão os 3 canais básicos de comunicação do programa de aplicação com o sistema ADABAS, ou seja, canal de instruções, canal de dados e canal de mensagens.

#### a) DADOS

O canal de dados para estabelecimento do fluxo de informações na comunicação do programa de aplicação com a interface ADABAS é mantido por intermédio de argumentos passados como parâmetros na chamada à subrotina. Estes parâmetros dizem respeito às áreas de dados que o programador deverá definir em seu programa destinados a:

- . Conter a descrição do formato dos dados (área de formato).

- . Armazenar o valor dos campos de acordo com o formato proposto (área-do-registro).

- . Armazenar a lista de números internos de seqüência dos registros (ISN) que satisfaçam ao critério de pesquisa especificado na instrução e que são retornados após a execução de uma operação de recuperação.

A definição destas áreas fica, portanto, sob responsabilidade do usuário, sendo que devem ser definidas como área de trabalho do programa (working storage).

O ADABAS recupera um bloco de registros do armazenamento físico em cada execução da macro CALL mas a interface com a linguagem hospedeira passa ao programa de aplicação somente os campos que foram requisitados (área-de-formato). Estes campos são apresentados no formato solicitado pelo programa que pode ser diferente do que consta na representação global.

Antes de acessar o banco de dados ADABAS, o programador deve possuir os detalhes de todos os campos e arquivos que o programa utilizará juntamente com as informações sobre o relacionamento entre os arquivos.

Conclui-se com isto que, o programador é responsável pela inicialização e organização da área de comunicação de dados na "working storage section" do programa, envolvendo, também, a indicação de todos os campos que serão utilizados.

Exemplificando o uso da comunicação de dados entre usuário e banco de dados poder-se-ia ter as seguintes construções na linguagem COBOL:



```

:
:
WORKING-STORAGE SECTION.
*-----
01 AREA-DE-FORMATO      PIC X(20) .
01 AREA-DO-REGISTRO    PIC X(20) .
01 AREA-DA-CONDICAO   PIC X(20) .
01 AREA-DO-VALOR      PIC X(20) .
01 AREA-DOS-ISNS      PIC X(256) .
01 BLOCO-DE-CONTROLE.
    02 LING-HOSP       PIC X VALUE 'C' .
        :
    02 CODIGO-COMANDO  PIC XX .
        :
    02 CODIGO-RETORNO  PIC S9(3) COMP-2 .
        :
PROCEDURE-DIVISION.
*-----
        :
*   MONTAGEM DOS CONTEUDOS DOS PARAMETROS
*
        :
CALL 'ADABAS'
    USING BLOCO-DE-CONTROLE, AREA-DE-FORMATO
        AREA-DO-REGISTRO, AREA-DA-CONDICAO
        AREA-DO-VALOR, AREA-DOS-ISNS.
        :

```

## b) INSTRUÇÕES

Instruções são fornecidas à interface de linguagem hospedeira através de um campo, contendo o código da função, que se localiza em uma das 6 áreas de comunicação passadas como parâmetro na macro CALL e que se denomina bloco de controle.

Para que a operação seja executada pelo sistema de gerência de banco de dados, a interface ADABAS deverá receber também, além do código da função, alguns critérios de seleção e valores associados a esses critérios. Estas informações estarão contidas em outros 2 argumentos, denominados área de pesquisa (search-buffer) e área de valor (value-buffer), que também serão passados como parâmetros na chamada à subrotina de interface.

Para a manipulação dos dados via os programas de aplicação o programador deverá, inicialmente, definir as suas áreas de entrada e saída, registros lógicos, variáveis e constantes de apoio e demais recursos necessários para estabelecer a comunicação com o banco de dados.

Uma vez definidas as áreas de dados, que serão usadas como argumentos de passagem de parâmetro na chamada da interface 'ADABAS', o usuário deverá codificar, nos pontos estratégicos do programa, as macros "CALL" correspondentes às funções que desejar executar.

O ADABAS tem capacidade de recuperar dados a nível de campo ao invés de a nível de registro permitindo, com isto, uma certa independência de dados.

A comunicação entre o programa e a subrotina 'ADABAS' é feita por meio de dois tipos de parâmetros: um bloco de controle e 5 áreas de controle (control buffers).

A chamada à subrotina segue o seguinte padrão na linguagem COBOL:

```
CALL 'ADABAS' USING
    <bloco-de-controle>, <área-de-formato>
    <área-de-registro>, <área-de-condição>
    <área-do-valor>, <área-dos ISNS>.
```

O bloco de controle é utilizado por todas as funções e os demais parâmetros são utilizados de acordo com as características de cada função, sendo que nem todos os parâmetros são necessários.

Os parâmetros são utilizados com as seguintes finalidades:

- . <bloco-de-controle> Define algumas informações de controle para o ADABAS como código do comando, linguagem hospedeira, número do arquivo, código de retorno, o ISN do registro, senha do usuário, comprimento das áreas e outras informações relevantes à execução de uma operação. Não será fornecido, aqui, o layout desta área pois ela é dependente da implementação.
- . <área-de-formato> Cadeia de caracteres que descreve o formato dos dados (visão local) com nome e tamanho de cada campo selecionado para a operação e, opcionalmente, espaços entre os campos.
- . <área-do-registro> Contém os valores dos campos segundo o formato especificado.
- . <área-da-condição> Especifica o critério de seleção dos registros e o formato dos valores contidos na <área-do-valor>. Cada condição de seleção envolvendo um acoplamento físico segue o seguinte padrão:
  - . Número que identifica o arquivo.
  - . Qualificação envolvendo campos chaves do arquivo (identificador + tamanho). As qualificações são do tipo:
    - . Condicionais(maior (GT), menor (LT), maior ou igual (GE), menor ou igual (LE) ...)

- . Booleanos (e (D), ou (O))
- . Intervalo de valores (de/até (S), exceto (N)).
- . <área-do-valor> Contém uma cadeia de valores a serem usados nas comparações.
- . <área-dos-ISNS> armazena a lista dos ISNS dos registros que satisfazem o critério de seleção.

Para ilustrar o uso dos parâmetros na seleção de registros, o caso de uma pesquisa de empregados do departamento de número 244001 com idade entre 20 a 25 anos, poderia ser exemplificado como segue.

O conteúdo colocado em <área-da-condição> e <área-do-valor> seria o seguinte:

```
<área-da-condição>: "/2/ED,6,D,EI,2,S,EI,2  
<área-do-valor> : 2440012025
```

Para complementar esta análise da linguagem de manipulação dos dados do ADABAS será apresentado um quadro com os principais comandos:

COMANDO	CÓD	FUNÇÃO
FIND NORMAL	S 1	- LOCALIZA REGISTROS CONFORME CRITÉRIO DE SELEÇÃO
<i>FIND SORT</i>	<i>S 2</i>	<i>- IDEM S 1 C/ ORDENAÇÃO</i>
FIND HOLD	S 4	- IDEM S 1 C/ BLOQUEIO DO 1º REGISTRO DA LISTA
FIND COUPLED	S 5	- LOCALIZA OS ISNS DE UM ARQUIVO ACOPLADOS A UM ISN ESPECÍFICO
COMP. ISN LIST	S 8	- UNIÃO / INTERSECÇÃO / COMPLEMENTAÇÃO SOBRE LISTA DE ISN
SORT ISN LIST	S 9	- CLASSIFICAÇÃO DE LISTA DE ISN
READ ISN	L 1	- LEITURA DO REGISTRO IDENTIFICADO PELO ISN
READ PHYSICAL SEQ	L 2	- LEITURA DO PRÓXIMO REGISTRO NA SEQÜÊNCIA FÍSICA
READ LOGICAL SEQ.	L 3	- LEITURA DO PRÓXIMO REGISTRO NA SEQÜÊNCIA LÓGICA
READ HOLD	L 4	- IDEM L 1 COM BLOQUEIO
READ P. HOLD	L 5	- IDEM L 2 COM BLOQUEIO
READ L. HOLD	L 6	- IDEM L 3 COM BLOQUEIO
READ FIELD DEF.	L 7	- LEITURA DAS DESCRIÇÕES DOS CAMPOS DE UM ARQUIVO
READ VALUES	L 9	- LEITURA DE VALORES ASSOCIADOS A UM DESCRITOR
ADD NEW RECORD	N 1	- INSERE NOVO REGISTRO
UPDATE RECORD	A 1	- ALTERA REGISTRO
UPDATE RECORD REL	A 4	- ALTERA REGISTRO LIBERANDO DO BLOQUEIO
DELETE RECORD	E 1	- ELIMINA REGISTRO
DEL. REC. RELEASE	E 4	- ELIMINA REGISTRO LIBERANDO O BLOQUEIO
OPEN	OP	- IDENTIFICAÇÃO DE NOVO USUÁRIO (ABERTURA DE ARQUIVOS)
CLOSE	CL	- FECHAMENTO DE ARQUIVOS

Figura 5.7 - Quadro de instruções DML do ADABAS

### c) MENSAGENS

Após a execução de cada macro CALL, o programador deverá verificar o código de resposta que vem definido no bloco de controle, recebido como parâmetro, a fim de analisar o seu conteúdo e saber se o comando foi bem sucedido ou não.

#### 5.3.3.2 Conversões

O sistema armazena os dados em um formato especificado pelo administrador do banco de dados, porém o usuário tem a liberdade de especificar formatos adversos quando da manipulação destes dados. Caso um programa de aplicação utilize os dados com outro formato, o ADABAS, desde que o programador indique o formato que está utilizando (nos padrões do sistema), fará a conversão para o formato de armazenamento.

Dependendo da implementação, o usuário terá a liberdade de especificar, na <área-de-formato>, mudanças no tamanho dos campos e também mudanças no formato [/DAV 77/]. Para este caso, o padrão de definição de formato é o seguinte:

```
... [<nro-char-branco>X,]<nome-campo>,[<tamanho>],[<formato>], ...
```

Em outras implementações, o usuário terá somente a liberdade de especificar mudanças no tamanho dos campos, além de espaços em branco entre os campos [/TSI 77/].

#### 5.3.3.3 Estruturas de controle de fluxo

Apesar do ADABAS permitir, através de determinados comandos, recuperar um conjunto de ISNS de registros que foram qualificados pelo critério de seleção, a transferên-

cia dos registros para a área de trabalho do programa deverá ser realizada considerando um registro de cada vez. A fim de que este fluxo de iteração para recuperação dos registros se materialize, cabe ao usuário codificar, em seu programa fonte, os comandos da linguagem hospedeira necessários para montar uma estrutura de controle da lógica do programa que permita especificar os procedimentos de recuperação dos registros individualmente a cada iteração.

A montagem de uma estrutura de controle de fluxo, está vinculada a um teste explícito do código de resposta do estado da operação retornado no bloco de controle que é recebido como parâmetro do comando "CALL".

#### 5.3.3.4 Considerações sobre proteção dos dados

No que se refere ao aspecto de segurança, o sistema ADABAS está dotado de facilidades de especificação de controle de acesso a nível de programa, arquivo e campo. Cada programa deve fornecer, por ocasião da abertura do banco de dados, uma senha (no bloco de controle) associada a qual existe um nível de segurança. O programa só terá acesso aos campos que tiverem nível de segurança inferiores ao nível de permissão do programa.

A facilidade de especificação de visões locais restringe também o uso dos dados às aplicações que possuem a autorização para a sua manipulação.

Nos aspectos de integridade, o ADABAS garante que os dados introduzidos no banco de dados são submetidos a uma verificação de formato assim que passarem pela área do registro (record buffer). O formato é comparado com aquele especificado na tabela de descrição dos campos e/ou área de formato especificada pelo usuário. Erros são notificados ao programa colocando o código de resposta apropriado no bloco de controle.

Há, também, oportunidades limitadas para o programador corromper o banco de dados através de enganos (mau uso de comandos ou valores mal armazenados) visto que nenhum dado estrutural é tornado disponível a ele durante a execução normal de seu programa. Os dados para acesso a registros, como listas invertidas, não podem ser modificados diretamente pelos programas de aplicação.

Como mecanismos de proteção pode-se considerar o fato de que o ADABAS oferece uma opção (status protection) que provoca a atualização física de um arquivo de transações (LOG) a cada comando de modificação; isto garante em caso de falhas da CPU, que as informações sobre comandos de modificação que foram logicamente bem sucedidos estarão gravados neste arquivo de transações. O arquivo de transações (LOG) conterá, portanto, as imagens anteriores e posteriores dos resultados alterados e também pontos de verificação (checkpoint). Para recuperação em caso de falha, o ADABAS oferece utilitários para restauração total ou parcial do banco de dados. Utilizando uma cópia do banco de dados e o arquivo de transações oferece as opções de REGENERATE, que desfaz as alterações a partir de um ponto de verificação, ou de BACKOUT, que desfaz as alterações até o ponto de verificação.

O ADABAS proporciona, também, facilidades que permitem ao usuário proteger os dados em um ambiente de atualização concorrente pela opção de bloqueio (hold) de um registro ou uma lista de registros.

#### 5.3.3.5 Considerações sobre homogeneidade

Analisando a homogeneidade entre a DML do ADABAS e as construções e comandos da linguagem hospedeira pode-se concluir somente que existe uma certa uniformidade no tratamento de chamada a interface S.G.B.D. quando comparada com



o tratamento de chamada a subrotinas externas convencionais. Em ambos os casos haverá passagem e/ou recepção de parâmetros que deverão ser devidamente manuseados pelos procedimentos internos do programa de aplicação.

Quanto à homogeneidade entre o tratamento da DML nas diferentes linguagens hospedeiras, pode-se afirmar que os dados e instruções que interagem com a DML serão tratados de acordo com as construções específicas de cada linguagem hospedeira e os padrões podem mudar de uma linguagem para outra.

#### 5.3.4 Aspectos de implementação do embutimento

##### 5.3.4.1 Descrição da forma de embutimento

O embutimento de instruções de manipulação de dados do ADABAS nas linguagens hospedeiras é implementado através de chamada a uma interface padrão de interpretação destas instruções. Esta interface de linguagem hospedeira, denominada "ADABAS", cuja comunicação com o programa de aplicação é materializada através do uso da macro CALL que passa os parâmetros adequados para execução dos comandos de manipulação (DML), é responsável por:

- . Analisar e interpretar os comandos ou instruções.

- . Tratar do registro lógico (visão local) que envolve a verificação dos campos solicitados, e o formato associado a cada campo.

- . Realizar as devidas conversões do formato definido no banco de dados para o padrão especificado para o programa de aplicação.

- . Montar os critérios de seleção do registro com

base no parâmetro recebido.

. Realizar o acesso ao núcleo do banco de dados recuperando os registros que satisfazem o critério de seleção.

. Extrair do registro selecionado os campos solicitados pelo usuário.

. Montar, na área de parâmetro, os campos extraídos dos registros recuperados conforme o layout e formato especificados.

. Caso algum problema for constatado, informar o código de erro.

Para o usuário, portanto, esta forma de embutimento é tratada como se fosse uma chamada a uma subrotina externa convencional passando, como parâmetro, os argumentos que serão usados como base para execução da tarefa por parte da subrotina externa.

#### 5.3.4.2 Considerações sobre a portabilidade da linguagem

A linguagem de manipulação de dados do ADABAS é tratada por uma interface padrão de linguagem hospedeira que está em contato com o programa de aplicação através da macro CALL.

Com isto conclui-se que o ADABAS poderia ser utilizado por outras linguagens hospedeiras que não as especificadas anteriormente (COBOL, PL/1, FORTRAN e ASSEMBLER), desde que esta linguagem tivesse, em suas construções básicas, a facilidade de invocar subrotinas externas através da macro CALL ou outro comando similar.

## 5.4 - O Sistema IMS

### 5.4.1 Caracterização do sistema

#### 5.4.1.1 Introdução

O IMS (Information Management System) é um S.G.B.D. [/DAT 82/, /DAV 77/, /FUR 82/ e /HEU 82/] produzido pela Companhia IBM (International Business Machines) para uso em computadores IBM 360/370.

Este sistema é um pacote do tipo DB/DC ("data base"/"data communication") possuindo, portanto, além das facilidades para definição e manipulação de informações do banco de dados, a versatilidade para a comunicação de dados através de terminais interativos.

#### 5.4.1.2 Estruturas de dados

O IMS é essencialmente um sistema de gerência de banco de dados que utiliza a abordagem hierárquica.

A linguagem de manipulação DL/1 (data language/1) sustenta estruturas de dados hierárquicos cada qual consistindo de um número de segmentos (grupos de campos).

Por intermédio de declarações especiais, do tipo índice, por exemplo, pode-se estabelecer ligações lógicas que são apontadores que permitem a simulação de relacionamentos do tipo não hierárquico. No entanto, a declaração das entidades deve sempre seguir o modelo hierárquico, podendo-se fazer referências às ligações lógicas dentro de uma hierarquia.

- Os componentes básicos de um S.G.B.D. IMS são grupos de campos (segmentos), banco de dados físicos e banco de dados lógicos. Estruturas de dados lógicos são defini

das especificando-se relacionamentos inter-segmentos. Os segmentos podem todos pertencer a um arquivo, tendo um particular método de acesso (B.D. físico) ou pode ser membro de diferentes arquivos (B.D. lógico). Permite, portanto, a definição de um banco de dados lógico derivado de vários bancos de dados físicos.

- Um banco de dados físico é um arquivo contendo um certo número de registros de um tipo particular (em IMS, arquivo é sinônimo de banco de dados).

- Um registro do banco de dados consiste em até 255 tipos de segmentos (numa conotação convencional, segmento corresponde ao conceito de registro sendo que constitui uma ocorrência de um determinado tipo de segmento (segment type = record type)) organizados em uma estrutura hierárquica, tipo árvore, tendo no máximo 15 níveis.

- A menor unidade lógica de dados neste sistema é o campo.

- Um registro do banco de dados possui um segmento raiz ("root") e um determinado número de segmentos dependentes. Cada segmento dependente pode ter somente um segmento físico "pai" ("parent") e pode ter diversos segmentos físicos "filhos" ("children"). O relacionamento entre dois segmentos situados em níveis adjacentes da árvore de definição do banco de dados é chamado pai-filho ("parent-child"). Dois segmentos de mesmo tipo, situados sob o mesmo pai, são chamados gêmeos ("twins"), sendo que um tipo de segmento pode ter diversos números de ocorrências ("physical twins").

- No banco de dados lógico podem ser especificados relacionamentos que não existem no banco de dados físico, o que permite a definição de várias visões diferentes para a mesma organização física [/FUR 82/].

#### 5.4.1.3 Linguagens hospedeiras

O usuário pode interagir com o sistema de gerência de banco de dados IMS através de programas de aplicação escritos em uma das suas linguagens hospedeiras como COBOL, FORTRAN, PL/1 e ASSEMBLER que referenciam o IMS por intermédio da utilização da macro CALL, para chamada de uma rotina padrão de interface que dependerá da linguagem hospedeira. Tem-se, por exemplo, a interface "CBLTDLI" para o COBOL e a "PLITDLI" para o PL/1.

No presente estudo do IMS será considerada, como linguagem alvo para descrição, a linguagem hospedeira PL/1.

#### 5.4.1.4 Estrutura funcional

A estrutura funcional do IMS pode ser generalizada através da figura 5.8.

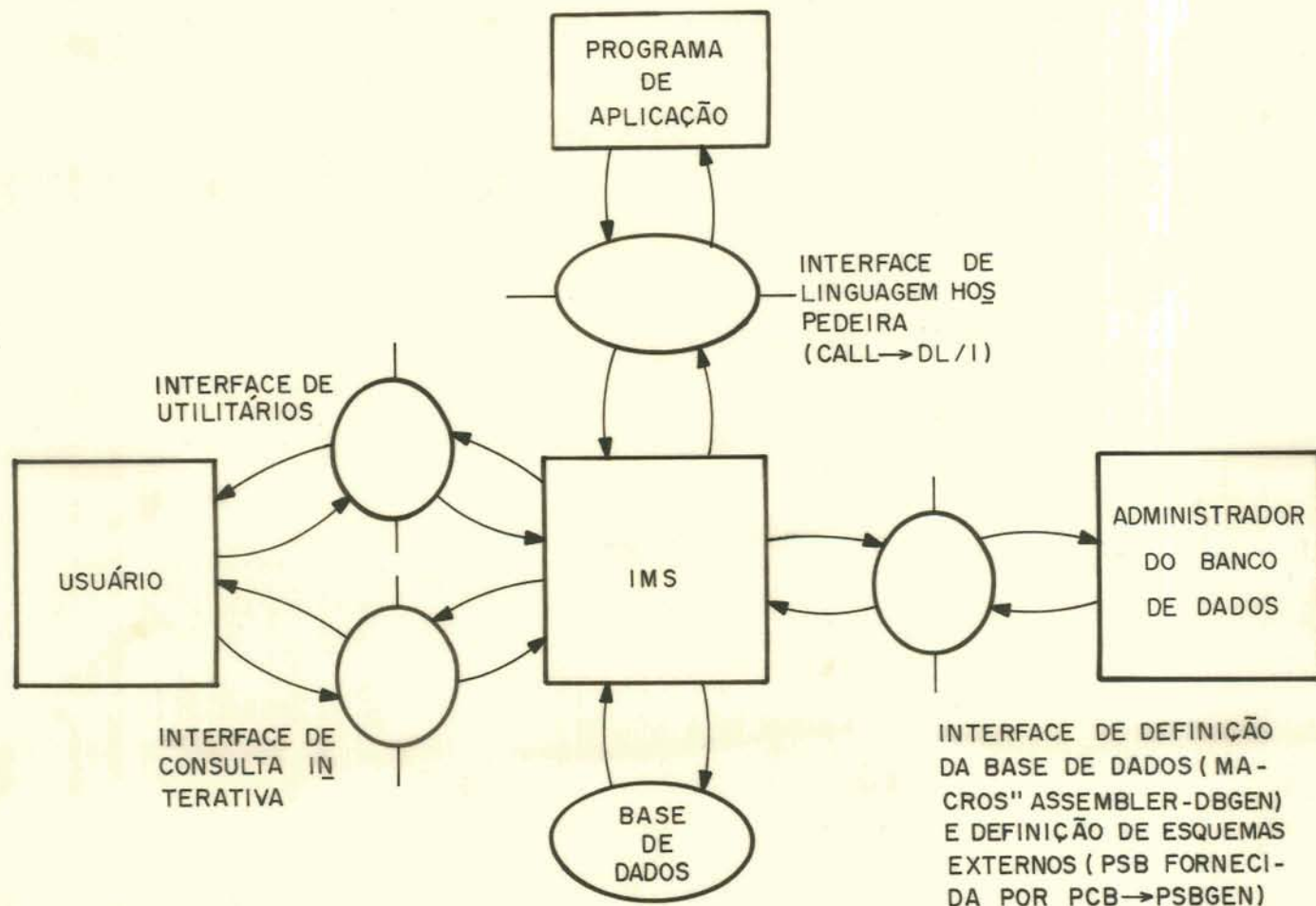


Figura 5.8 -Arquitetura funcional do IMS

Nesta análise serão consideradas somente as interfaces de linguagem hospedeira e as de definição do banco de dados e definição de esquemas externos.

#### 5.4.2 Interface de definição dos dados no sistema

##### 5.4.2.1 Definição da base de dados

A linguagem de definição dos dados do IMS (DDL) consiste de uma série de macroinstruções assembler que descrevem o banco de dados nos aspectos físicos, lógicos e os

esquemas externos. De uma forma geral, pode-se classificar os comandos desta linguagem em 3 categorias distintas:

. Comandos destinados a definição de um banco de dados físico (DBD).

. Comandos destinados à definição de um banco de dados lógico ou visão local dos dados para um banco de dados físico (PCB).

. Comandos destinados à definição de esquemas externos que determinada aplicação tem do conjunto de banco de dados sobre o qual vai operar (PSB).

Inicialmente será visto como são usadas as 2 primeiras categorias de comandos. A última categoria de comandos será tratada especificamente no item "Definição dos esquemas externos".

#### Banco de dados físico

Cada banco de dados físico, no IMS, é definido, juntamente com o seu mapeamento para armazenamento e indicação do método de acesso, por uma descrição do banco de dados (DATA BASE DESCRIPTION - DBD).

A definição é feita em termos de campos e segmentos (grupos de campos) e pode haver até 255 tipos de segmentos definidos para cada banco de dados.

Os comandos fonte que compõem a descrição do banco de dados são montados e o código objeto resultante é armazenado em uma biblioteca do sistema, como bloco de controle, para posterior uso. A descrição do banco de dados é analisada pelo utilitário DBDGEN que verifica a sua correção e caso esteja correta gera uma tabela de definição que será armazenada e posteriormente usada pelo IMS no atendimento aos comandos das linguagens de manipulação.

A definição tem início com um comando DBD que contém informações sobre o nome do banco de dados, o método de acesso empregado, opcionalmente uma senha e outras informações que dependem do método de acesso.

DBD NAME = <nome-DB>, ACCESS = <método de acesso>

Os métodos de acesso podem ser:

- . HSAM (Hierarchical Sequential Access Method) - Acesso em uma seqüência hierárquica por contigüidade física.

- . HISAM (Hierarchical Indexed Sequential Access Method) - Acesso indexado seqüencial ao segmento raiz e acesso seqüencial aos segmentos dependentes.

- . HDAM (Hierarchical Direct Access Method) - Acesso ao segmento raiz é por cálculo de endereço (HASH).

- . HIDAM (Hierarchical Indexed Direct Access Method) - Acesso direto a um segmento raiz através de um índice associado.

A definição segue com um comando DATASET que contém informações relativas ao nome do data-set (posteriormente referenciado por comandos JCL) e estrutura de armazenamento como dispositivo, blocagem, tamanho do registro, etc.

DATASET DD1 = <nome data-set>, DEVICE = <nro dispositivo>, BLOCK= <tam.bloco>, RECORD = <tam.registro>, ...

Uma descrição de segmento é precedida por um comando SEGM que indicará o nome do segmento, nome do segmento pai, tamanho em bytes, regras de modificação e outras informações adicionais.



```
SEGM NAME = <nome do seg>, PARENT = <nome do pai>,
          BYTES = <tam. do seg>, RULES = <regras>, ...
```

A ordem em que um comando SEGM aparece na definição determina a posição do segmento na estrutura hierárquica resultante.

Para definir relacionamentos lógicos é usado o comando LCHILD onde é informado o nome do segmento pai e nome do banco de dados além do nome do ponteiro.

```
LCHILD NAME = (<nome seg>, <nome B.D.>),
            INDEX = <nome pont.>
```

Existe, portanto, a possibilidade de estabelecer relacionamentos entre segmentos que existem em diferentes bancos de dados físicos.

Os campos são definidos por comandos do tipo FIELD e que poderão ser referenciados por um programa de aplicação. Neste comando será especificado o nome do campo, indicação se é chave primária (SEQ), o tamanho em bytes, a posição dentro do segmento (pode haver superposição de campos) e opcionalmente o tipo do campo (hexa - X, Decimal - P, Alfanumérico - C).

```
FIELD NAME = (<nome campo>, [SEQ]), BYTES = <tamanho>
            START = <pos.início>, TYPE = <tipo de dado>
```

Para definir campos indexados que utilizam índices secundários (arquivo invertido) tem-se o comando XDFLD onde é informado o nome atribuído ao índice e nome do campo que será usado como índice secundário.

```
XDFLD NAME = <nome ind>, SRCH = <nome campo sec>
```

A definição é finalizada com os comandos DBDGEN,

FINISH e END.

A título de ilustração, a definição de parte do banco de dados exemplo, é mostrada abaixo:

```

DBD          NAME = PESSOAL, ACCESS = HDAM
DATASET DD1 = PESSDS, DEVICE = 3330
SEGM        NAME = DEPT, BYTES = 35
FIELD       NAME = (DEPNRO,SEQ), BYTES=5, START=1
FIELD       NAME = DEPNOOME, BYTES=30, START=6
SEGM        NAME = EMP, PARENT = DEPT, BYTES = 46
:
DBGEN
FINISH
END

```

#### Banco de dados lógico

Cada banco de dados lógico, no IMS, é definido por um bloco de comunicação do programa (Program Communication Block - PCB). O PCB faz o mapeamento entre o banco de dados lógico e o físico correspondente. Cada PCB contém detalhes sobre:

- . Tipo de banco de dados (DB ou DC).
- . Nome do banco de dados físico, base para a definição.
- . Restrições de acesso ou opções de processamento (I - Inserção, R - Alteração, D - Exclusão, G - Leitura) através da cláusula "PROCOPT".
- . Tamanho da área para concatenação das chaves primárias manipuladas (raiz até posição corrente) segundo o caminho hierárquico.
- . Nomes de índices secundários e nome dos segmen-

tos que farão parte do banco de dados lógico (comando SENSEG).

. Se posicionamento simples ou múltiplo é permitido.

A estruturação de definição de um banco de dados lógico segue a seguinte sintaxe:

```
PCB TYPE = <tipo de B.D.>, DBDNAME = <nome BD físico >,
        PROCOPT = <opções proc>, KEYLEN = <tam.chaves>
```

```
SENSEG NAME = <nome seg>, PARENT = <nome seg pai>
        PROCOPT = <opções proc>, INDICES = <nome ind.sec>
```

#### 5.4.2.2 Definição de esquemas externos

Uma vez definido o banco de dados lógico através dos blocos de comunicação do programa (PCB) é possível especificar os esquemas externos disponíveis às diversas aplicações.

Portanto, uma visão do usuário, usada pelos programas de aplicação, pode envolver vários bancos de dados lógicos, cada qual especificado por um PCB, e é definida por um bloco de especificação do programa (PSB).

A definição de um PSB está intimamente relacionada com a definição de um PCB. Portanto não se pode traçar uma fronteira entre eles pois os comandos de definição de um complementam os comandos do outro.

Daí, se conclui que um PSB contém uma lista de segmentos que estão disponíveis a um programa através de um determinado PCB (segmentos sensitivos - SENSEG).

Um PSB é definido através do comando PSBGEN que especifica a linguagem hospedeira em que o programa de aplicação será escrito e o nome pelo qual será referenciado o

PSB.

Como sintaxe completa de definição de PSB's tem-se:

```
PCB
SENSEG
SENSEG
PCB
:
PSBGEN LANG = <ling hosp>, PSBNAME =<nome do PSB>
END
```

Além desta definição, sob responsabilidade do administrador de dados, o programador especifica os segmentos necessários a seu programa de aplicação quando faz a chamada ao IMS através dos argumentos de pesquisa de segmentos (SSA).

#### 5.4.3 Aspectos funcionais do embutimento

##### 5.4.3.1 Comunicação do usuário com o banco de dados

A área de comunicação do usuário com o sistema de gerência de banco de dados IMS aparece esquematizada na figura 5.9.

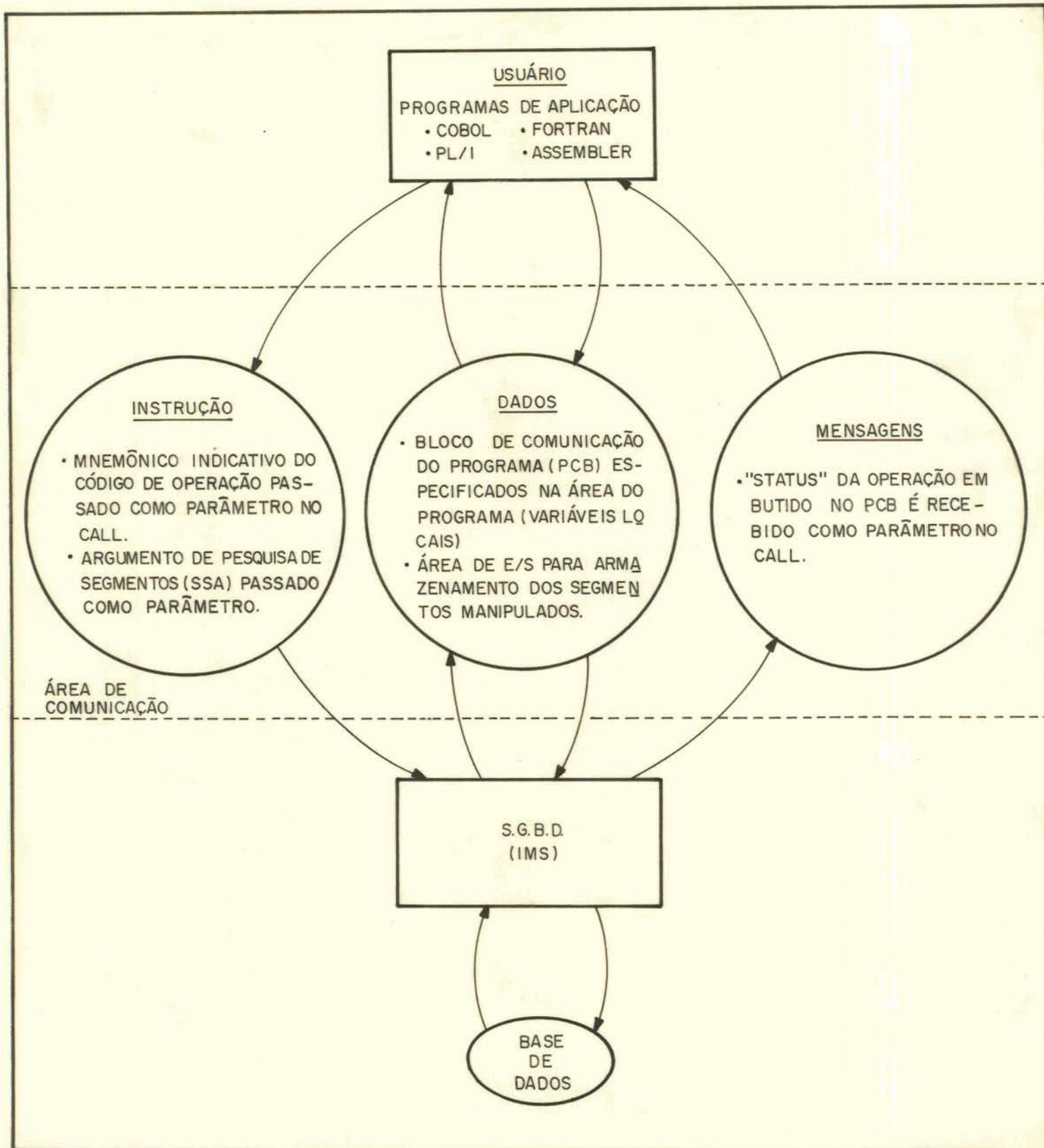


Figura 5.9 - Canais de comunicação programas de aplicação e IMS

A comunicação do usuário e o sistema IMS, em uma interface de linguagem hospedeira é realizada através da macro "CALL" de chamada a uma das rotinas que realizam a interface. Cada linguagem hospedeira tem uma rotina de interface associada.

Os três canais de comunicação, ou seja, instruções, dados e mensagens são materializados no programa de aplicação através dos próprios parâmetros da macro "CALL".

O programador é responsável pela organização das comunicações entre programa e DL/1 via a "Working Storage Section" do programa. Isto envolve a colocação de todos os campos referenciados nas chamadas ao DL/1 junto com uma área de comunicação PCB (PCB MASK). Fica, portanto, sob sua responsabilidade a especificação de:

- . A área de trabalho e comunicação de cada programa.
- . Os comandos da linguagem de manipulação de dados.
- . Uma lista de segmentos necessários ao programa (segment search arguments).
- . Os comandos da linguagem hospedeira necessários para controlar o fluxo de dados entre o programa e DL/1 e para verificar se cada comando DL/1 tem sido bem sucedido.

#### a) DADOS

Para o canal de dados o programador deverá especificar um bloco de comunicação do programa (PCB) que conterá, entre outras informações, o mapeamento do banco de dados lógico para o banco de dados físico.

O conteúdo de um PCB pode ser sintetizado da seguinte forma:

CONTEÚDO	TAM. BYTES
Nome do banco de dados	8
Número do nível hierárquico	2
Status da operação	2
Opções de processamento	4
Nome do segmento corrente	8
Tamanho da chave corrente	4
Número de segmentos sensitivos	4
Chaves concatenadas correntes	n

Na linguagem COBOL, por exemplo, este bloco de comunicação é declarado na "Linkage Section" do programa.

O endereço do PCB constitui um dos parâmetros de uma chamada ao IMS através do CALL. Há necessidade de especificar um PCB para cada banco de dados lógico acessado. Em PL/1 os blocos de comunicação são declarados como uma lista de variáveis do tipo pointer como pode ser visto no exemplo a seguir:

```

:
:
DECLARE
        ENDPCB      POINTER;
DECLARE
        1 PCB       BASED (ENDPCB) ,
        2 NOME DB   CHAR (8) ,
        2 NIVELSEG CHAR (2) ,
:

```

Além do PCB, faz parte do canal de dados a especificação de uma área de entrada e saída para armazenar os segmentos lidos em uma operação de leitura ou os segmentos entregues ao IMS em uma operação de inserção ou alteração.

Estas poderiam ser declaradas como mostra o exemplo, em PL/1:

```

:
DECLARE EMP-ES      CHAR(54),
      1 EMP DEFINED EMP-ES,
      2 EMP-NRO     CHAR(5),
      2 EMP-NOME    CHAR(30),
      2 EMP-DATAADM CHAR(6),
      2 EMP-IDADE   CHAR(2);

```

Verifica-se, portanto, que o programador necessita estar ciente de todos os campos, segmentos e estruturas hierárquicas que existem no banco de dados lógico a ser acessado.

#### b) INSTRUÇÕES

As instruções são passadas a interface de linguagem hospedeira por intermédio de um parâmetro da macro CALL, que será chamado de comando da DML. Este parâmetro é uma variável do programa que receberá um mnemônico indicativo do código da operação a ser executada.

As instruções são baseadas em uma sintaxe muito rudimentar e artificial (baixo nível) e fazem sempre referência a um registro de cada vez.

A sintaxe de chamada à interface, na linguagem COBOL é a seguinte:

```

CALL 'CBLTDL1' USING
      <nro-parâmetros>,<comando-DML>
      <nome-da-PCB>,<area-ent-sai>
      <lista-argumentos>

```

Cada parâmetro representa um endereço que será



passado ao IMS, sendo que devem ser codificados numa ordem física preestabelecida e possuem a seguinte finalidade:

- . <nro-parâmetro> Especifica o número de parâmetros no call (parâmetro é obrigatório somente para a linguagem PL/1).
- . <comando-DML> Indica o código da função que se deseja executar.
- . <nome-da-PCB> Nome da área de comunicação do programa e o IMS no formato que já foi apresentado anteriormente.
- . <área-ent-sai> Área de entrada e saída a ser utilizada para recepção e transmissão de segmentos.
- . <lista-argumentos> Lista de argumentos de pesquisa utilizados para identificar os segmentos a serem recuperados (um SSA por cada tipo de segmento pertencente ao caminho de acesso). O formato de um SSA segue o seguinte padrão:

<nome-segmento> [<código-comando>] [( <condição> )]

- . <nome-segmento> É o nome de um tipo de segmento pertencente ao caminho de acesso.
- . <código-comando> É a especificação de operações alternativas sobre os segmentos do caminho de acesso. Ex: F indica que deve ser tomado o primeiro filho sob um segmento.
- . <condição> Conjunto de expressões de comparação (campo do segmento, operador de comparação e um valor) ligados pelos operadores booleanos e/ou.

Em vista do IMS adotar o modelo de dados hierárquico e considerando que uma hierarquia estabelece um caminho de acesso predefinido aos segmentos que compõem a árvore de pesquisa, o sistema deverá manter a posição corrente do cursor que aponta para o segmento que está sendo proces-

sado. Esta informação é mantida no PCB. Por conseguinte, o programador deverá, também, ter conhecimento sobre a concepção de posicionamento.

Uma pesquisa sobre uma hierarquia de segmentos segue o caminho de acesso pré-definido (pré-ordem), conforme pode ser visto nas figuras 5.10a e 5.10b.

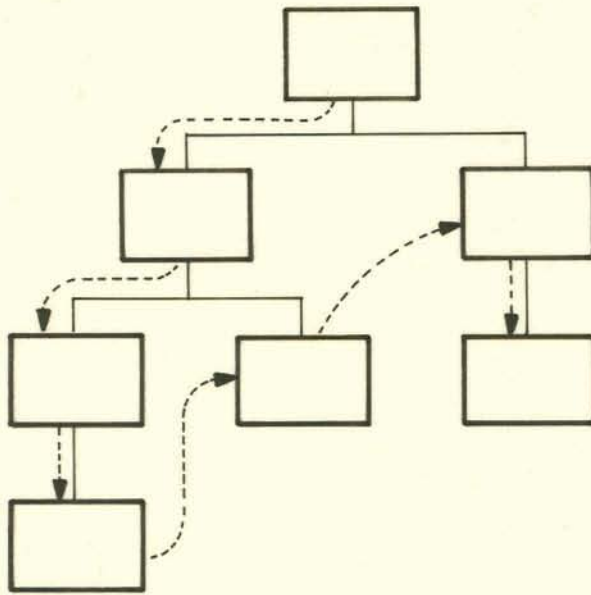


Figura 5.10a - Ordem de pesquisa nos tipos de segmentos

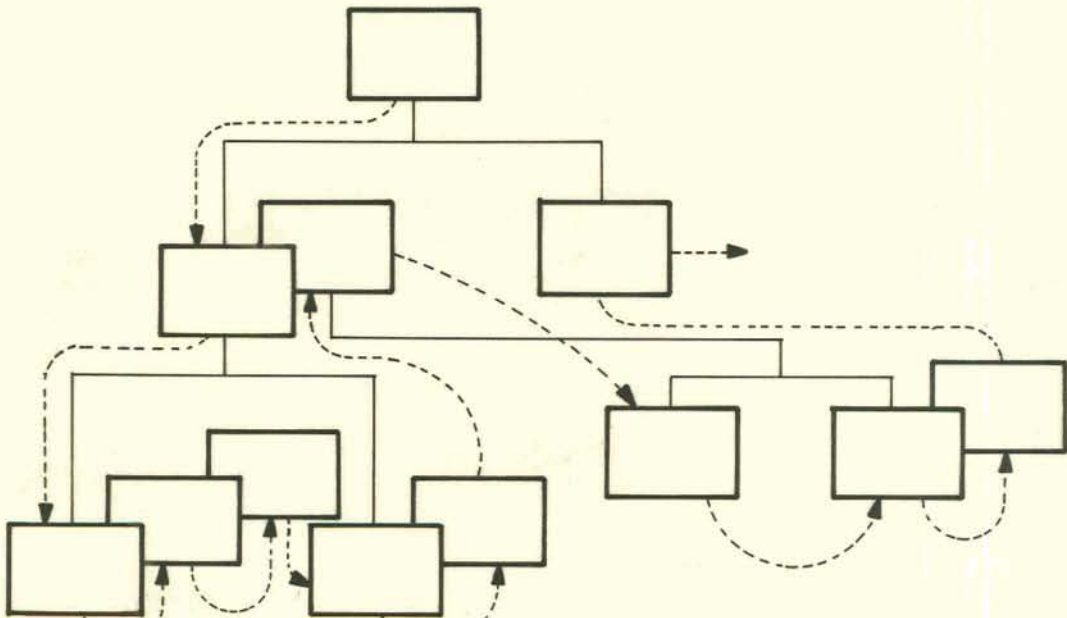


Figura 5.10 b - Ordem de pesquisa em ocorrências de segmentos

Além da posição corrente de segmento processado (posicionamento simples), o IMS admite opcionalmente especificar posicionamento múltiplo através do qual é mantida a posição de cada segmento no caminho hierárquico sendo processado, permitindo que diferentes tipos de segmentos sobre um mesmo pai sejam processados em paralelo.

O efeito de uma instrução, portanto, depende da posição corrente de um cursor no momento da execução.

A posição corrente permite ao DL/1 processar comandos do tipo GET NEXT.

Aliado às instruções o programador especifica também um parâmetro que conterá um argumento de pesquisa de segmento (SSA - Segment Search Argument). O uso de SSAs é dependente do tipo de comando DML especificado, podendo ser usado para:

- . Identificar um determinado segmento pelo nome.
- . Recuperar um segmento condicionado ao valor de um dos campos.
- . Permitir uma variação dos comandos DML especificados nos CALL a ser aplicada a um determinado segmento.

Em um programa PL/1 pode-se declarar as instruções e as SSAs como mostra o exemplo a seguir:

```
⋮  
DECLARE  
    GU      CHAR(4) INIT('GU  '),  
    GN      CHAR(4) INIT('GN  '),  
    GNP     CHAR(4) INIT('GNP ');  
  
⋮  
DECLARE  
    1 MASC-SSA  STATIC UNALIGNED,  
    2 NOMESEG   CHAR(8) INIT('      '),  
    2 ABREPAR   CHAR(1) INIT('('),  
    2 ARGPEAQ   CHAR(20)  
    2 FECHAPAR  CHAR(1) INIT('(');
```

Para completar os comentários sobre as instruções o quadro a seguir sintetiza os comandos que podem ser usados no IMS.

TIPO COMANDO	COMANDO	CÓD.	FUNÇÃO
CONTROLE	CHECKPOINT	CHKP	- CRIA UM REGISTRO DE LOG
	RESTART	XRST	- RESTAURA ÁREA DO PROGRAMA CONTIDA NO LOG
	LOG	LOG	- PERMITE QUE O USUÁRIO COLOQUE INFORMAÇÕES NO LOG
	GETSCD	GSCD	- RECUPERA INFORMAÇÕES SOBRE O DIRETÓRIO DO IMS E DA TABELA DE DESCRIÇÃO DO B.D.
	STATISTICS	STAT	- OBTENÇÃO DE ALGUMAS ESTATÍSTICAS
RECUPERAÇÃO	GET UNIQUE	GU	- RECUPERA SEGMENTO ESPECIFICADO NO SSA
	GET NEXT	GN	- RECUPERA O SEGMENTO QUE SEGUE O SEGMENTO CORRENTE
	GET NEXT WITHIN PARENT	GNP	- RECUPERA O SEGMENTO QUE SEGUE O SEGMENTO CORRENTE MAS SOMENTE PARA O CORRENTE SEGMENTO PAI
MODIFICAÇÃO	INSERT	ISRT	- INSERE NOVO SEGMENTO
	DELETE	DLET	- EXCLUI SEGMENTO E SEUS DESCENDENTES
	REPLACE	REPL	- ALTERA CONTEÚDO DOS CAMPOS DE UM SEGMENTO

Figura 5.11: Quadro de instruções DML do IMS

OBS.: Para os comandos de recuperação existe uma modalidade que permite o bloqueio dos segmentos recuperados (hold). O código para estes comandos fica sendo, respectivamente, GHU, GHN, GHNP.

Todos comandos CALL envolvendo atualização e exclusão precisam ter sido precedidos por um comando de recuperação que terá deixado o segmento de interesse na área de entrada/saída.

#### c) MENSAGENS

O canal de mensagens é proporcionado através de um código de "status" da operação que vem embutido no bloco de comunicação recebido como um dos parâmetros da chamada ao IMS. Este código pode ser testado no programa de aplicação para verificar se a operação resultou bem sucedida.

Os valores que podem ser devolvidos pelo IMS no "status" da operação são, por exemplo:

- '00' - sucesso na operação
- 'GE' - fim do tipo de segmento
- 'GB' - final do banco de dados.

#### 5.4.3.2 Conversões

No IMS, não é feita nenhuma validação de formato dos campos, embora seja oferecida a possibilidade de especificação de tipo de campo. Por conseguinte nenhum mecanismo de conversão está implementado no sistema.

Em vista disso o usuário deve estar ciente do layout de cada segmento manipulado pelo programa, bem como do formato de cada campo que compõe este segmento, pois deverá ter definido, na área de trabalho do programa, uma máscara correspondente ao segmento usado, a fim de ter acesso ao campo desejado. Se esta máscara estiver mal definida, resultados errôneos podem ser obtidos.

#### 5.4.3.3 Estruturas de controle de fluxo

Fica sob responsabilidade do usuário codificar em seu programa fonte, através de comandos da linguagem hospedeira, a lógica de controle do fluxo de iteração para recuperação dos registros. A cada nova iteração ("loop") um registro é recuperado e um teste do código de "status" da operação indicará quais os procedimentos a serem seguidos dentro de uma estrutura de controle de iteração adequada. O controle da iteração estará, portanto, vinculado a um teste explícito do código de "status" da operação retornado.

#### 5.4.3.4 Considerações sobre proteção dos dados

Do ponto de vista de segurança pode-se salientar o fato de que algumas restrições de acesso podem ser impostas a nível de um banco de dados lógico completo ou a nível de um determinado segmento sensível manipulado pelo programa de aplicação. Estas restrições são definidas quando da especificação do bloco de comunicação do programa (PCB) e são do tipo "opções de processamento" (inserção, alteração, exclusão e leitura).

A facilidade de definição de PSB's restringe também o uso de determinadas partes do banco de dados aos programas que estão autorizados para tal.

Além disso, existe no IMS, quando da utilização dos módulos para comunicação de dados (DC) via terminal interativo, que não foi discutido nesta análise, um mecanismo de controle de senhas para condicionar o acesso de determinado terminal a um banco de dados específico.

A integridade pode ser vista, no IMS, como uma característica muito limitada e deficitária pois nenhuma verificação de formatos nos dados é realizada, embora que ne-

nhum dos dados estruturais, armazenados fisicamente com os segmentos, é tornado disponível ao programador, evitando, assim, corrupções,

No banco de dados físico a exclusão de um segmento resulta na exclusão de todos os seus dependentes. Já no banco de dados lógico isto pode ser controlado por intermédio das "opções de processamento" especificadas no bloco de comunicação do programa (PCB).

No aspecto de proteção, facilidades são proporcionadas que permitem que dados sejam protegidos num ambiente de atualização concorrente através do uso de comandos do tipo "get-hold" na linguagem de manipulação antes de realizar uma modificação. Além disso tem, também, um mecanismo de "logging & recovery", proporcionado a partir de determinados comandos de controle da linguagem de manipulação de dados DL/1.

#### 5.4.3.5 Considerações sobre homogeneidade

A homogeneidade entre construções da DML e as construções da linguagem hospedeira reside no fato da linguagem hospedeira dispensar um tratamento semelhante, ao da chamada de qualquer subrotina convencional externa, quando invocar a interface do S.G.B.D. através da macro CALL. Internamente, no ambiente do programa de aplicação, as instruções DML serão manipuladas em concordância com os demais parâmetros passados e/ou retornados do IMS. Estes parâmetros serão analisados e manuseados pelos comandos da linguagem hospedeira.

Em se tratando de comparar, no entanto, este tratamento, entre as diversas linguagens hospedeiras, pode-se concluir que ele vai depender dos padrões de cada linguagem hospedeira específica. Isto porém não traz maiores implicações pois, como foi visto, as construções da DML são manipu



ladas pelos próprios comandos de cada linguagem hospedeira.

#### 5.4.4 Aspectos de implementação do embutimento

##### 5.4.4.1 Descrição da forma de embutimento

A forma de realizar o embutimento de instruções de manipulação de dados do IMS (linguagem DL/1) nas linguagens hospedeiras é extremamente simples. É implementado através de chamada de interface padrão de interpretação destas instruções. Neste tipo de embutimento tudo se passa como se fosse a chamada (CALL) de uma subrotina externa convencional. Nenhum comando específico da linguagem DL/1 está embutido na linguagem hospedeira COBOL, FORTRAN, PL/1 ou ASSEMBLER.

Já que o uso do DL/1 não envolve comandos embutidos no programa do usuário, compilar um programa de linguagem hospedeira não é diferente do processo normal.

Durante o processo de execução, o IMS analisa as funções, nomes e endereços referenciados pelos parâmetros no comando CALL com auxílio das definições do banco de dados armazenadas em uma biblioteca.

O processo de embutimento e comunicação com o módulo de manipulação dos dados (DL/1) pode ser visualizado através da figura 5.12.

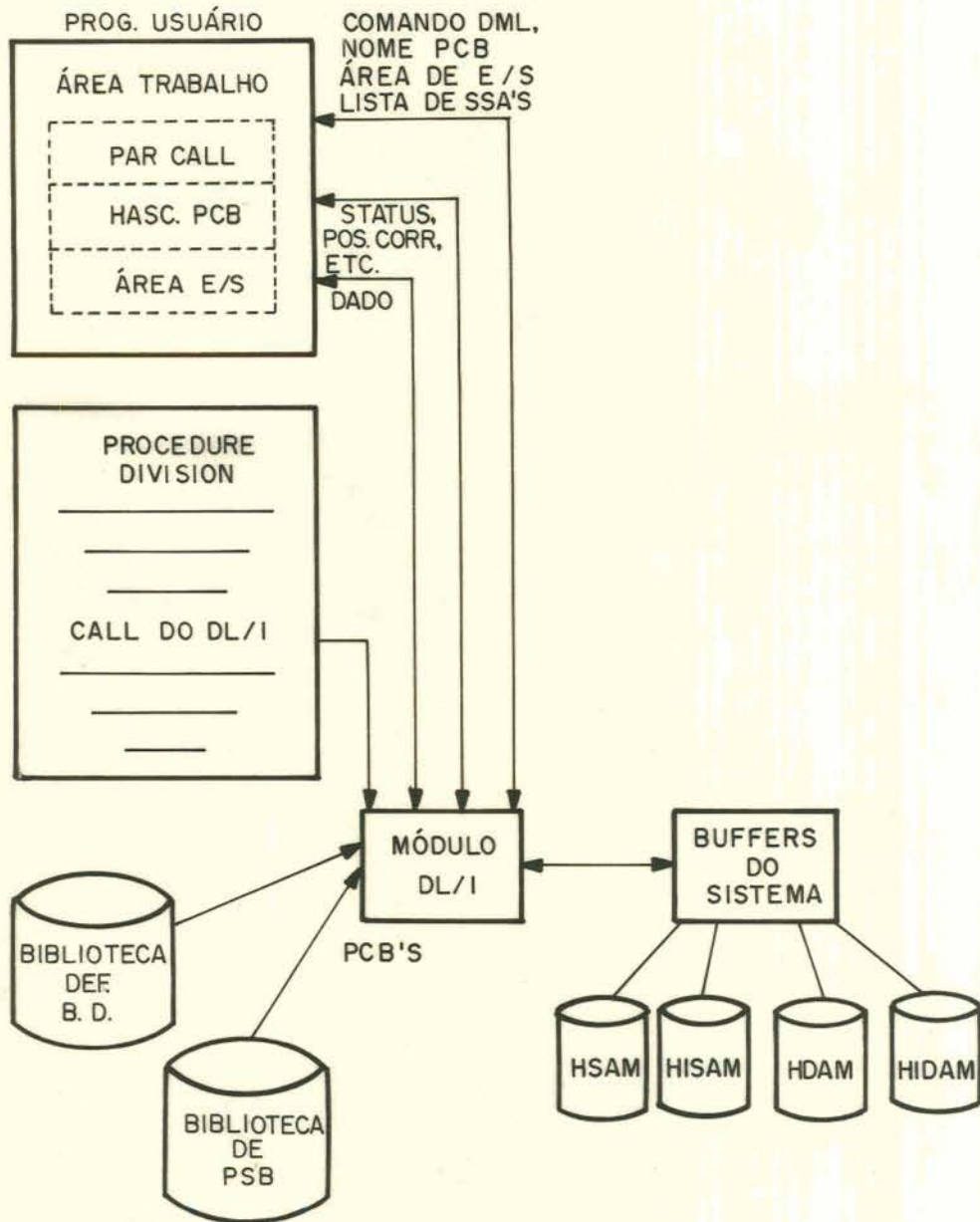


Figura 5.12 - O IMS embutido em linguagem hospedeira

#### 5.4.4.2 Considerações sobre a portabilidade da linguagem

Teoricamente a linguagem DL/1 poderia ser embutida em outras linguagens hospedeiras desde que esta linguagem possua a facilidade de chamada de rotinas externas. Ressalva-se, no entanto, que o IMS foi concebido para operar sobre os sistemas operacionais que se alojam em ambientes de máquinas IBM 360/370 e a adaptação deste sistema para o ambiente operacional de máquinas de outros fabricantes consome significantes recursos de projeto, desenvolvimento e implementação.

## 5.5 O Sistema IDMS

### 5.5.1 Caracterização do sistema

#### 5.5.1.1 Introdução

O IDMS (Integrated Database Management System) é um S.G.B.D. [/COH 78/, /DAV 77/] baseado no modelo DBTG ("Data Base Task Group") da CODASYL. Foi originalmente desenvolvido pela companhia americana B.F. GOODRICH e posteriormente produzido pela CULLINAME CORP, para computadores IBM 360/370, ICL 2900/1900 e DEC PDP-11.

#### 5.5.1.2 Estruturas de dados

Para a estruturação do banco de dados o IDMS utiliza a abordagem em rede, implementando um subconjunto de especificações elaboradas pelo DBTG da CODASYL.

Os relacionamentos (set type) são do tipo 1:n e não têm atributos associados.

- O relacionamento entre registros, ou SET na terminologia do IDMS, é basicamente uma lista encadeada composta de um registro mestre (owner) e um ou mais registros membros (member) sendo que um registro pode ser mestre de mais de um SET e pode, também, participar como membro de mais de um SET (estrutura em rede).

- Relacionamento hierárquico entre campos no registro podem ser estabelecidos através de um número de nível na descrição do registro.

- Os apontadores que resultam dos relacionamentos são guardados nos próprios registros das entidades podendo ser estabelecidos 3 tipos de apontadores para um SET. São eles: ao próximo (next), ao anterior (prior), ao registro

mestre (owner). Com isto a navegação na estrutura de dados se torna mais otimizada quando for necessário regressar ou conhecer o registro mestre.

#### 5.5.1.3 Linguagens hospedeiras

O IDMS pode ser usado pelos programas de aplicação em duas modalidades distintas. Através de linguagens hospedeiras com comandos da DML embutidos numa modalidade de extensão desta linguagem ou através de linguagens hospedeiras com chamada a uma subrotina que fará a interface com o S.G.B.D.

Na primeira modalidade, o sistema IDMS está disponível para as linguagens COBOL, PL/1 e FORTRAN sendo que para a segunda modalidade, pode também ser usado por programas escritos em FORTRAN, ASSEMBLER e outras linguagens procedurais, se bem que com maior interferência do programador na tarefa de especificação das áreas de comunicação.

No estudo e descrição das características do IDMS será considerado, como linguagem alvo, o COBOL na modalidade de extensão das construções originais da linguagem (implementado através da precompilação).

#### 5.5.1.4 Estrutura funcional

A estrutura funcional do sistema IDMS está disposta de acordo com o que pode ser visto na figura 5.13.

O assunto será estendido com maiores detalhes no decorrer do texto da análise feita sobre este sistema.

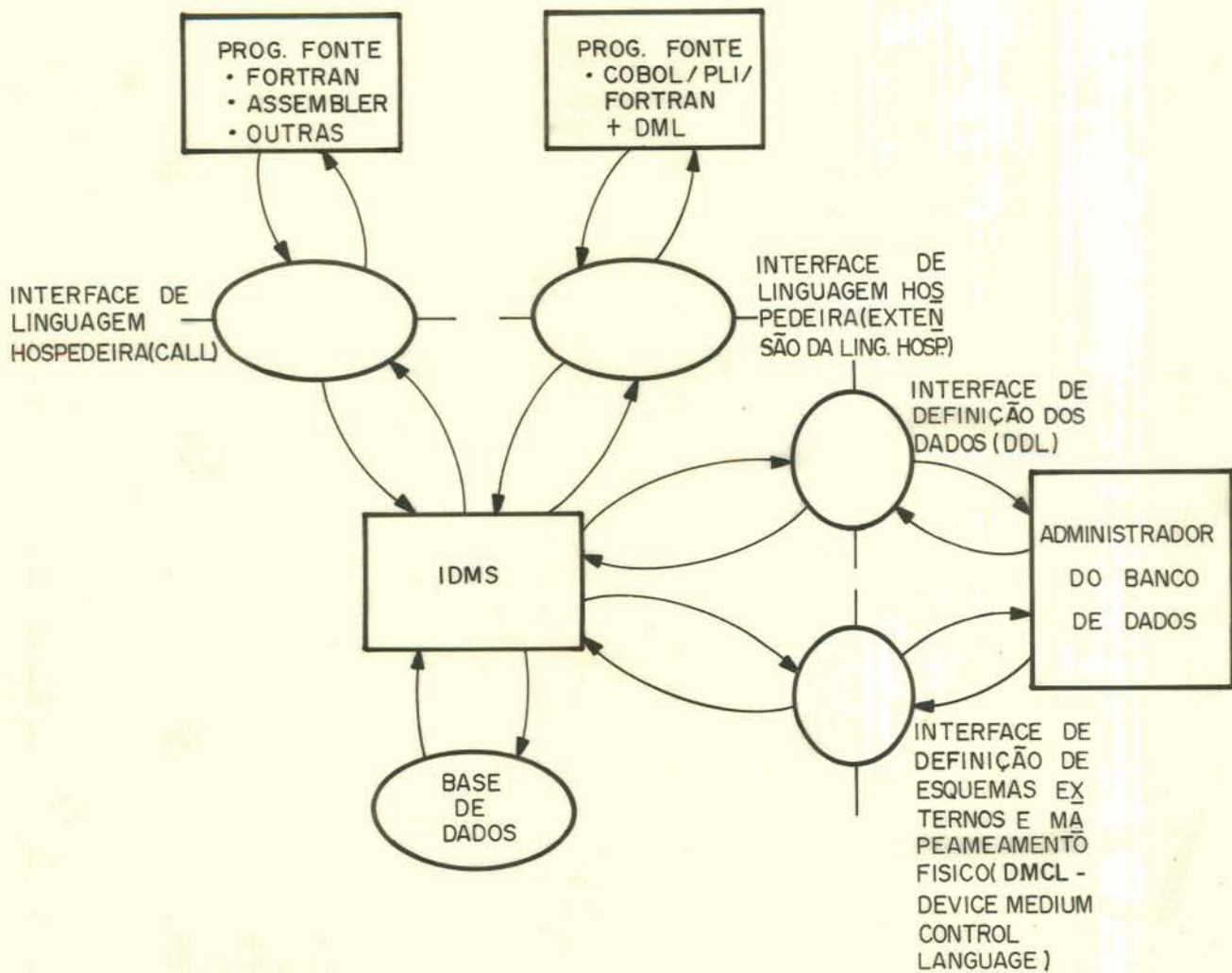


Figura 5.13 - Arquitetura funcional do IDMS

## 5.5.2 Interface de definição dos dados no sistema

### 5.5.2.1 Definição da base de dados

A descrição do banco de dados é realizada usando-se uma linguagem própria para esta finalidade conhecida como DDL (Data Definition Language) sendo que a descrição é realizada em termos de descrição de esquema (banco de dados físico), áreas, registros e sets.

Além disso, para atender as necessidades dos programas de aplicação, a especificação de esquemas externos ("subschemas") cujos detalhes serão discutidos mais adiante, também são incluídos na definição dos dados.

A tarefa fica sob a responsabilidade do administrador do banco de dados que fará a definição dos dados codificando os comandos adequados na DDL.

Uma vez codificados, os comandos da DDL são processados pelo compilador "IDMS SCHEMA DDL" e colocados num diretório de dados (Data Directory) que manterá, além da definição dos dados, o relacionamento lógico, esquemas externos do banco de dados e dados necessários ao controle do mapeamento lógico-físico.

Do ponto de vista do usuário, o banco de dados será constituído de um certo número de áreas lógicas especificadas pela DDL.

A sintaxe da DDL segue padrões muito similares aos da linguagem COBOL.

Não é intenção mostrar aqui uma descrição detalhada da sintaxe mas analisar, em rápidas pinceladas, a sua composição básica.

Normalmente a definição dos dados pela DDL segue a seguinte ordem de especificação:

- . SCHEMA DESCRIPTION (que informa o nome do banco de dados e opcionalmente a data de criação e outras informações adicionais);

- . FILE DESCRIPTION (que indica o nome externo do arquivo e o tipo de dispositivo de armazenamento);

- . AREA DESCRIPTION (determinará as áreas necessárias para o arquivo);

- . RECORD DESCRIPTION (mostrará a descrição do registro com todos campos que a ele estarão vinculados);

- . SET DESCRIPTION (definirá o relacionamento entre 2 ou mais registros sendo que fisicamente será conseguido através de ponteiros armazenados nos registros).

Cada conjunto de registros (entidade) recebe, além do nome, um número de identificação (cláusula "RECORD ID IS") e uma indicação do modo de armazenar as ocorrências do registro (cláusula "LOCATION MODE IS"). Os modos de armazenar as ocorrências são:

- . DIRECT - o sistema decidirá onde armazenar.

- . CALC - usa uma função de "HASH" do sistema IDMS para atribuir um endereço ao registro, baseado no campo chave.

- . VIA - coloca registros na proximidade de outros membros em um relacionamento (SET) especificado, podendo-se até especificar a distância.

Além disso uma rotina de tratamento de exceções pode ser especificada como estando associada a operações sobre a entidade [/DAV 77/].

Na descrição dos campos do registro, é usada a nomenclatura COBOL, sendo permitidos os tipos caracter, binário, decimal e ponto flutuante, podendo-se atribuir um va



lor inicial e grupos repetitivos.

Na definição do relacionamento (SET) é especificado, além do nome do relacionamento, também a ordem que será usada para processamento seqüencial de membros de uma ocorrência (cláusula "ORDER IS"), o nome do registro mestre, o nome do registro membro e a forma como os registros menores serão inseridos (classes de SET).

As classes de SET podem ser:

- . Mandatory Automatic
- . Mandatory Manual
- . Optional Automatic
- . Optional Manual

onde:

. AUTOMATIC - o IDMS insere um membro automaticamente no SET quando o registro é armazenado.

. MANUAL - o usuário deverá inserir um membro no relacionamento (SET).

. MANDATORY - inserção é permanente no relacionamento.

. OPTIONAL - permite que o usuário remova um membro do relacionamento.

A ordem dos registros no SET pode ser:

- . First - novo membro logo após o mestre.
- . Last - novo membro no fim da lista.
- . Next - novo membro logo após o registro corrente.
- . Prior - novo membro antes do registro corrente.
- . Ascending/Descending - é mantida a ordenação na lista.

A título de ilustração será mostrado como o banco de dados de exemplo pode ser definido:

```

SCHEMA DESCRIPTION.
SCHEMA NAME IS PESSOAL.
AUTHOR. ADMINISTRADOR.

:
FILE DESCRIPTION.
FILE NAME IS ARQPES ASSIGN TO 'BD/PESSOAL'
                                DEVICE TYPE IS DISK.

AREA DESCRIPTION.
AREA NAME IS AREA-1.

RECORD DESCRIPTION.
RECORD NAME IS DEPT.
    RECORD ID IS Ø1.
    LOCATION MODE IS CALC USING ...
    WITHIN AREA-1.
    CALL ROTINA-DB ON ERROR DURING STORE.
Ø2 DEP-NRO    PIC 9(6) COMP-2.
Ø2 DEP-NOME  PIC X(30) DISPLAY.
Ø2 DEP-LOCAL PIC X(30) DISPLAY.

:
RECORD NAME IS EMP.
    RECORD ID IS Ø2.
    LOCATION MODE IS VIA ...
    WITHIN AREA-1.
Ø2 EMP-NRO   PIC 9(6) COMP-2.
Ø2 EMP-NOME  PIC X(30) DISPLAY.

:
SET DESCRIPTION.
SET NAME IS LOTACAO.
ORDER IS NEXT.
OWNER IS DEPT.
    NEXT DBKEY POSITION IS ...

```

```
MEMBER IS EMP.
      NEXT DBKEY POSITION IS ...
      MANDATORY AUTOMATIC.
```

#### 5.5.2.2 Definição de esquemas externos

Os programas de aplicação operam sobre esquemas externos designados, na terminologia IDMS, subesquemas. Estes são especificados usando o "SUBSCHEMA DDL" que é compilado e o resultado armazenado no diretório de dados para ser ligado ao programa objeto do usuário em tempo de execução.

Um subesquema consiste de 1 ou mais áreas do banco de dados físico (SCHEMA). A visão do usuário destas áreas pode se restringir a um registro particular; campos do registro ou a relacionamentos lógicos (SET).

Chaveamentos (PRIVACY LOCK) podem ser especificados num esquema externo e são usados para restringir o acesso a uma área, registro ou relacionamento.

Cada subesquema está constituído de 2 partes. Uma "IDENTIFICATION DIVISION" que identifica o subesquema em termos de nome e áreas lógicas necessárias e uma "DATA DIVISION" onde são especificados a descrição dos registros e relacionamentos que fazem parte deste subesquema e restrições de acesso.

Um exemplo da definição de um subesquema pode ser visto abaixo:

```
SUBSCHEMA IDENTIFICATION DIVISION.
      SUBSCHEMA NAME IS USUARIO1 OF SCHEMA NAME PESSOAL.
      DEVICE-MEDIA NAME IS ...
      :
```

SUBSCHEMA DATA DIVISION.

AREA SECTION.

COPY AREA-1 AREA.

PRIVACY LOCK FOR EXCLUSIVE UPDATE IS "YES".

:

RECORD SECTION.

COPY DEPTO RECORD

PRIVACY LOCK FOR DELETE IS "NO".

Ø1 EMP

PRIVACY LOCK FOR MODIFY IS "YES".

Ø2 EMP-NRO PIC 9(6) COMP-2.

:

SET SECTION.

COPY LOTACAO SET

PRIVACY LOCK FOR DISCONNECT IS "NO".

:

### 5.5.3 Aspectos funcionais do embutimento

#### 5.5.3.1 Comunicação do usuário com o banco de dados

A área de comunicação do usuário com o sistema de gerência de banco de dados é mostrada na figura 5.14.

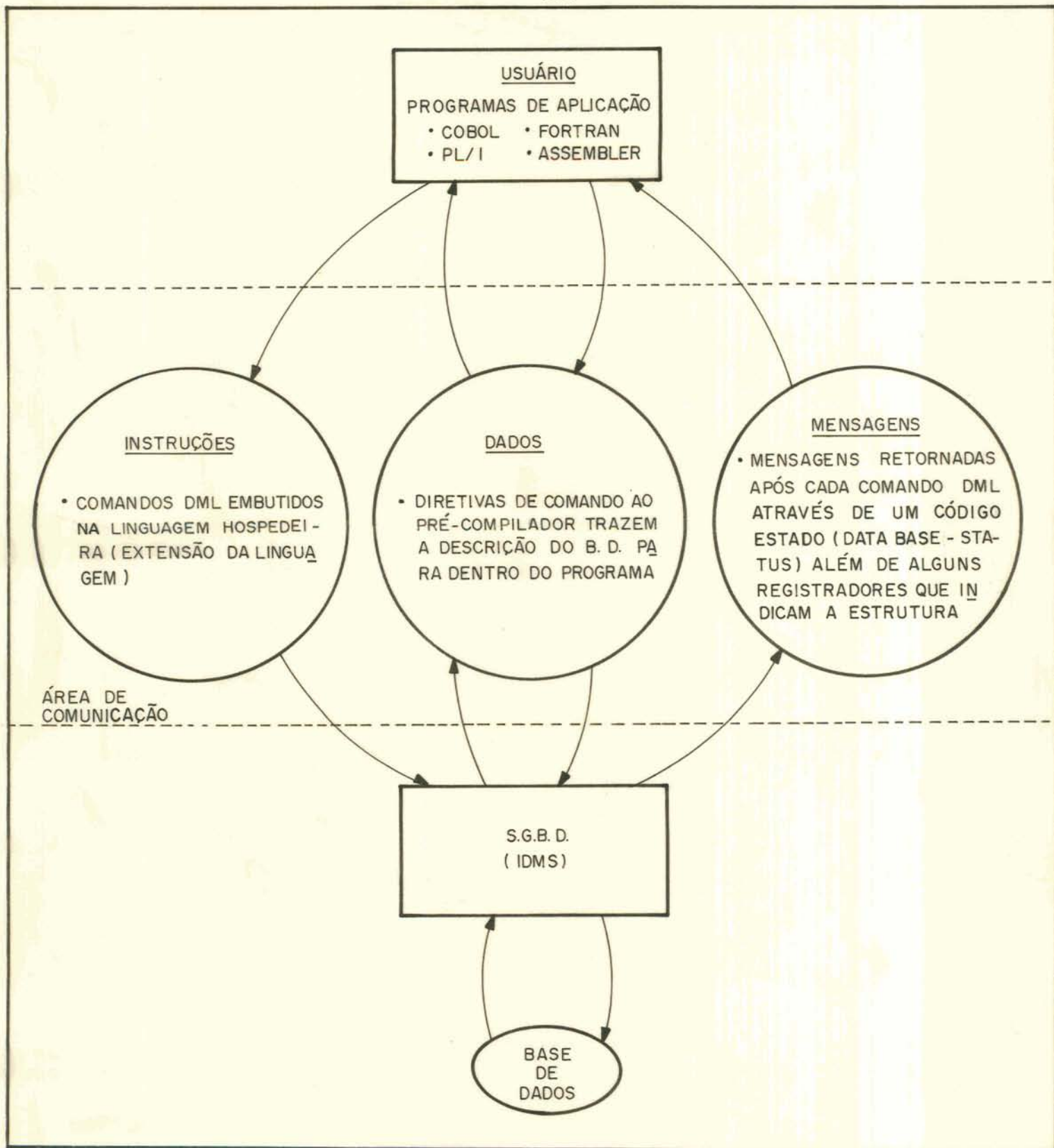


Figura 5.14 - Canais de comunicação programas de aplicação e IDMS

## a) DADOS

Cada programa que usa o banco de dados tem associado a ele uma área de comunicação de entrada/saída de registros, conjunto de variáveis previamente declaradas (que podem ser consideradas como palavras reservadas) com valores atribuídos pelo sistema, tais como códigos de erros (database-status), nome do último registro acessado (DB-record-name), etc.

O acesso ao banco de dados pode, somente, ser feito através de um subesquema previamente definido.

O usuário deverá codificar, na DATA DIVISION de seu programa uma SCHEMA SECTION na qual especifica o subesquema particular necessário a este programa. Com isto o processador DML (precompilador) inclui automaticamente na WORKING STORAGE SECTION do programa, a partir do diretório de dados do sistema (data directory) a descrição dos registros (layout B.D.) e áreas de controle.

Esta especificação é feita no seguinte formato:

```
DATA DIVISION.
SCHEMA SECTION.
DB <nome visão-local> WITHIN <nome-BD>
```

Com isto será trazido para dentro do programa toda a descrição dos registros que compõem o subesquema do usuário.

## b) INSTRUÇÕES

O programador se comunica com o banco de dados através da codificação de comandos DML em pontos apropriados da PROCEDURE DIVISION ou equivalente de seu programa. Estes comandos são posteriormente interpretados pelo processador DML (precompilação) e convertidos em comandos da linguagem hospedeira.

Ex.:

```

      :
      :
PROCEDURE DIVISION.
      :
      :
      FIND FIRST DEPTO RECORD OF LOTACAO SET
      :
      :

```

Os registros são recuperados segundo o protocolo "um-de-cada-vez" não havendo a possibilidade de obter um conjunto de registros através de uma única instrução ou comando.

A consulta aos conjuntos de dados (DATA-SET) é feita por intermédio de comandos que indicam explicitamente o percurso pelas entidades através dos relacionamentos.

Para assinalar os registros correntemente em uso, o IDMS utiliza marcas ou cursores ("Currency Indicators" data base keys) que podem ser de 4 tipos:

- . CURRENT OF PROGRAM (RUN-UNIT) - indica o registro mais recentemente recuperado com sucesso pela execução dos comandos FIND/OBTAIN ou STORE.

- . CURRENT OF RECORD TYPE - indica o último registro que foi "current of program" para um tipo particular de registro.

- . CURRENT OF SET - é o registro no set correspondente ao mais recente "current of program". Esta marca aparecerá em todos os sets nos quais o registro participa.

- . CURRENT OF AREA - é a área em que está o registro marcado pelo "current of program".

Estes cursores permitirão a navegação sobre os re

gistros do banco de dados e é de real interesse que o programador conheça bem os conceitos que foram mencionados para que possa minimizar o "overhead" no processamento.

Para complementar os comentários sobre a manipulação dos dados no IDMS, será mostrada uma relação sucinta dos principais comandos. Esta relação aparece na figura 5.15.

A sintaxe completa de cada comando não será alvo desta descrição.



TIPO	COMANDO	FUNÇÃO
CONTROLE	READY (OPEN)	- INICIALIZA AS ÁREAS RELATIVAS AO PROGRAMA COM O TIPO DE ACESSO DESEJADO.
	FINISH (CLOSE)	- LIBERA AS ÁREAS USADAS PELO PROGRAMA
	IF	- TESTA OCORRÊNCIA DE UM REGISTRO EM UM RELACIONAMENTO OU SE É MEMBRO DE UM RELACIONAMENTO.
RECUPERAÇÃO	FIND	- LOCALIZA E RECUPERA REGISTROS DE ACORDO COM UM CRITÉRIO DE SELEÇÃO. EXISTEM DIVERSAS MODALIDADES PARA ESTE COMANDO. POR EXEMPLO: <ul style="list-style-type: none"> <li>◦ FIND &lt;REG&gt; DB-KEY IS &lt;ID&gt; ( DIRETO )</li> <li>◦ FIND CURRENT &lt;REG&gt; ( CORRENTE )</li> <li>◦ FIND NEXT &lt;REG&gt; WITHIN &lt;SET&gt;( VIA SET )</li> <li>◦ FIND OWNER WITHIN &lt;SET&gt; ( REG. PAI )</li> </ul>
	GET	- TRAZ O REGISTRO CORRENTE PARA A ÁREA DE TRABALHO DO PROGRAMA
	OBTAIN	- COMBINA O FIND + GET. TEM A MESMA SINTAXE DO FIND
	ACCEPT	- PERMITE A RECUPERAÇÃO DOS CURSORES E COLOCÁ-LOS EM VARIÁVEIS DO PROGRAMA
MODIFICAÇÃO	ERASE	- PERMITE REMOVER REGISTROS DE UM RELACIONAMENTO E DO BANCO DE DADOS
	CONNECT	- INTRODUZ UM REGISTRO COMO MEMBRO DE UM SET
	DISCONNECT	- CANCELA A PARTICIPAÇÃO DE UM REGISTRO EM DETERMINADO SET
	STORE	- INSERE UM REGISTRO NO BANCO DE DADOS E EM TODOS OS SEUS SET'S AUTOMÁTICOS
	MODIFY	- PERMITE A ALTERAÇÃO DE CAMPOS DE UM REGISTRO DO BANCO DE DADOS

Figura 5.15 : Quadro de instruções DML do IDMS

Maiores detalhes sobre os comandos e sua sintaxe podem ser vistos na bibliografia [ /DAV 77/ ].

### c) MENSAGENS

Após cada comando DML executado, o programador deverá examinar o campo que contém o código de estado (data base-status) retornado do sistema IDMS para determinar se a operação foi concluída com sucesso ou não.

```
IF DATABASE-STATUS = ...
```

Além disso, para saber a que estrutura a mensagem se refere, existem registradores especiais que podem ser analisados. Por exemplo:

```
DB-REALM-NAME  
DB-SET-NAME  
DB-RECORD-NAME
```

### 5.5.3.2 Conversões

O usuário recebe, automaticamente, a descrição dos dados em seu programa nos padrões de cada linguagem hospedeira quando invocar uma diretiva do precompilador para este fim. O precompilador realizará as adaptações necessárias para que a descrição seja apresentada nos padrões da linguagem hospedeira.

### 5.5.3.3 Estruturas de controle de fluxo

No IDMS as iterações devem ser especificadas utilizando comandos adequados da linguagem hospedeira. Os registros sendo recuperados individualmente, um de cada vez, nas operações seriais de recuperação dos registros armazenados no banco de dados é necessário realizar um teste explí-

cito do conteúdo do código de estado (database-status) a fim de controlar o fim da iteração.

#### 5.5.3.4 Considerações sobre proteção dos dados

Com referência aos aspectos de segurança, pode-se considerar que as técnicas de subesquemas implementadas no sistema IDMS, auxiliam no sentido de que o usuário terá acesso somente aos dados que realmente tem direito. Um programa de aplicação somente acessa o banco de dados através de seu subesquema e estes podem ser definidos considerando somente os dados necessários àquela aplicação, impedindo o acesso aos demais dados.

Quanto à integridade, para prevenir os dados armazenados no banco de dados contra exclusões indevidas, o IDMS controla este procedimento através de um mecanismo de autorização denominado "PRIVACY LOCKS", especificados a nível de registro. Além disso o comando ERASE tem uma opção seletiva que previne que os registros não sejam deletados de outros relacionamentos que não o especificado. Contudo, cuidado é necessário quando registros são excluídos do banco de dados, particularmente quando o subesquema em uso não contém todos os relacionamentos em que o registro participa.

Além disso, um dado estrutural, que é armazenado com os registros não é tornado disponível ao programador. O IDMS somente transfere os registros lógicos que têm sido especificados em um subesquema específico.

Existe, no entanto, uma restrição no que se refere à integridade. O IDMS, normalmente, não realiza nenhuma verificação de formato dos dados manipulados. Esta verificação no entanto pode ser programada pelo usuário e incluída através das opções e "procedures", existentes no sistema, que permitem ao usuário final escrever rotinas específicas.

Em relação aos aspectos de proteção existe, no sistema IDMS, a possibilidade de contornar problemas causados por atualizações concorrentes especificando uma maneira apropriada de uso das áreas no "subschema area section" do esquema externo, ou seja, especificando acesso exclusivo para atualização ou recuperação. No caso que um usuário estiver lendo um registro enquanto outro usuário estiver atualizando-o, informações obsoletas provavelmente serão recuperadas do sistema. Para evitar isso o usuário pode especificar um modo exclusivo que permite uma utilização com exclusividade do registro corrente.

#### 5.5.3.5 Considerações sobre homogeneidade

Como foi observado pela análise das características do IDMS, existe uma uniformidade sintática muito grande entre este S.G.B.D. e a linguagem COBOL. Pode ser, portanto, constatada a homogeneidade entre os comandos DML do IDMS e os comandos da linguagem COBOL a tal ponto que podem representar uma extensão natural das construções da linguagem.

Realmente é fácil compreender esta uniformidade de sintaxe entre o IDMS e o COBOL visto que o IDMS é baseado e usa a terminologia do DBTG da CODASYL que foi responsável, também, pelo desenvolvimento da metodologia e padrões que deram origem à linguagem COBOL.

Homogeneidade análoga pode também ser atribuída para a linguagem PL/1 onde o próprio precompilador usado para esta linguagem realizará a adequação dos pequenos detalhes que diferenciam as construções desta linguagem.

Para as linguagens FORTRAN e ASSEMBLER, por sua vez, a diferença já se torna mais significativa mas, cabe a uma interface de precompilação a responsabilidade de produzir os ajustes necessários.

#### 5.5.4 Aspectos de implementação do embutimento

##### 5.5.4.1 Descrição da forma de embutimento

A DML do IDMS foi originalmente projetada para que a interface com a linguagem hospedeira (na ocasião o COBOL ANS) fosse via um precompilador que recebesse os comandos da DML embutidos no programa de aplicação, interpretando-os e gerando as chamadas de subrotinas necessárias. Foi projetado, portanto, como uma extensão do COBOL ANS sendo que a sintaxe dos comandos é usada de forma similar. Para o programador COBOL, acostumado com as construções desta linguagem, a tarefa de manipular os comandos DML do IDMS torna-se, com isto, muito facilitada, visto que ele, simplesmente, adiciona as declarações e comandos DML à "data division" e "procedure division" de seu programa.

Posteriormente, foram desenvolvidas, também, interface de precompilação para as linguagens PL/1 e FORTRAN, nos mesmos princípios da precompilação do COBOL.

Tem-se, portanto, uma forma de embutimento onde o programa do usuário consiste de comandos DML embutidos na linguagem hospedeira que são passados inicialmente pelo precompilador DML específico. Este obtém o nome do subesquema através da declaração "DB". Isto permite que o diretório de dados seja acessado para obter a completa descrição do subesquema para o programa. Esta descrição é então colocada na "WORKING STORAGE SECTION" do programa.

A validade dos comandos DML é verificada e o precompilador traduz cada comando em uma chamada única do IDMS por intermédio do comando CALL e coloca um asterisco (para linguagem COBOL) no comando DML original, para que se torne um comentário no programa fonte.

A saída do precompilador DML é um programa fonte na linguagem hospedeira que pode ser compilado por um compi

lador padrão.

A figura 5.16 mostra todo este processo.

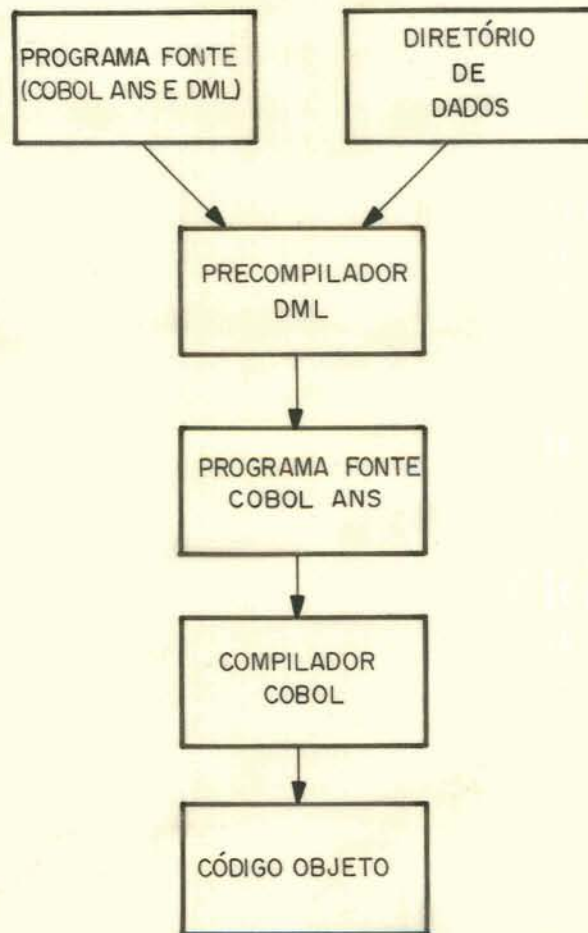


Figura 5.16 - O IDMS embutido em linguagem hospedeira

Além da forma de embutimento baseada em precompilação do programa fonte, outras linguagens hospedeiras podem ter uma interface direta com o IDMS através da chamada a subrotina externa por intermédio do comando CALL ou outro comando similar.

#### 5.5.4.2 Considerações sobre a portabilidade da linguagem

Existe uma certa dificuldade em implementar um embutimento em uma linguagem hospedeira que não as já disponíveis atualmente. A tarefa envolveria o desenvolvimento de um novo precompilador específico para a nova linguagem hospedeira com tratamento de todas as suas estruturas lógicas, como definição de registros e campos nos padrões desta linguagem, além de adequar, também, os comandos DML à nova sintaxe.

## 5.6 O Sistema DMS II

### 5.6.1 Caracterização do sistema

#### 5.6.1.1 Introdução

O DMS II (Data Management System II) é um S.G. B.D. [/BUR 77/, /BUR 78/, /FUR 82/] desenvolvido pela Burroughs Corporation, principalmente para os computadores da série B6000 e B7000, tendo, também, versões simplificadas para os modelos B1700 e B1800.

#### 5.6.1.2 Estruturas de dados

O sistema DMS II tem a sua sintaxe baseada na proposta DBTG ("Data Base Task Group") da Codasyl [/TAY 76/] e utiliza a abordagem em rede para estruturação dos seus dados, se bem que o próprio fornecedor do sistema faz, também, referência a uma abordagem hierárquica [/BUR 78/].

A estrutura básica do banco de dados é composta por conjunto de dados (DATA-SETS), como sendo uma coleção de registros; conjunto de índices (SETS) sobre todo o conjunto de dados e subconjuntos (SUBSETS) sobre parte do conjunto de dados.

Podem ser especificados vários SETS e SUBSETS para um mesmo DATA-SET possibilitando o uso de várias chaves de acesso. Os SETS podem conter atributos, ou seja, parte da informação dos registros pode ser armazenada nos SETS (especificando a cláusula DATA).

É possível especificar a localização de registros de entidades, sem a existência dos SETS ou SUBSETS, através de rotinas de endereçamento a partir das chaves de acesso ("ACCESS" para "DIRECT DATA-SET").



O DMS II permite declarar DATA-SETS dentro de outros ("EMBEDDED DATA-SETS"), o que atribui ao sistema certas características da abordagem hierárquica, devendo ser declarado um SET para cada DATA-SET embutido. Podem ter diversos níveis de embutimento, como nos modelos hierárquicos.

É possível montar relacionamentos cruzados, declarando-se um SUBSET relativo a um DATA-SET "A" dentro de um outro DATA-SET "B" disjunto do primeiro (relacionamentos não hierárquicos).

Permite relacionamentos do tipo 1:n (relacionamentos hierárquicos) e, também do tipo m:n (relacionamentos não hierárquicos).

#### 5.6.1.3 Linguagens hospedeiras

O DMS II, em sua linguagem de manipulação de dados, é usado nos programas de aplicação como uma extensão de uma linguagem hospedeira, estando, atualmente, disponível nas linguagens COBOL, ALGOL e PL/1.

A linguagem alvo para a presente descrição será o COBOL.

#### 5.6.1.4 Estrutura funcional

A estrutura funcional do sistema DMS II pode ser vista na figura 5.17.

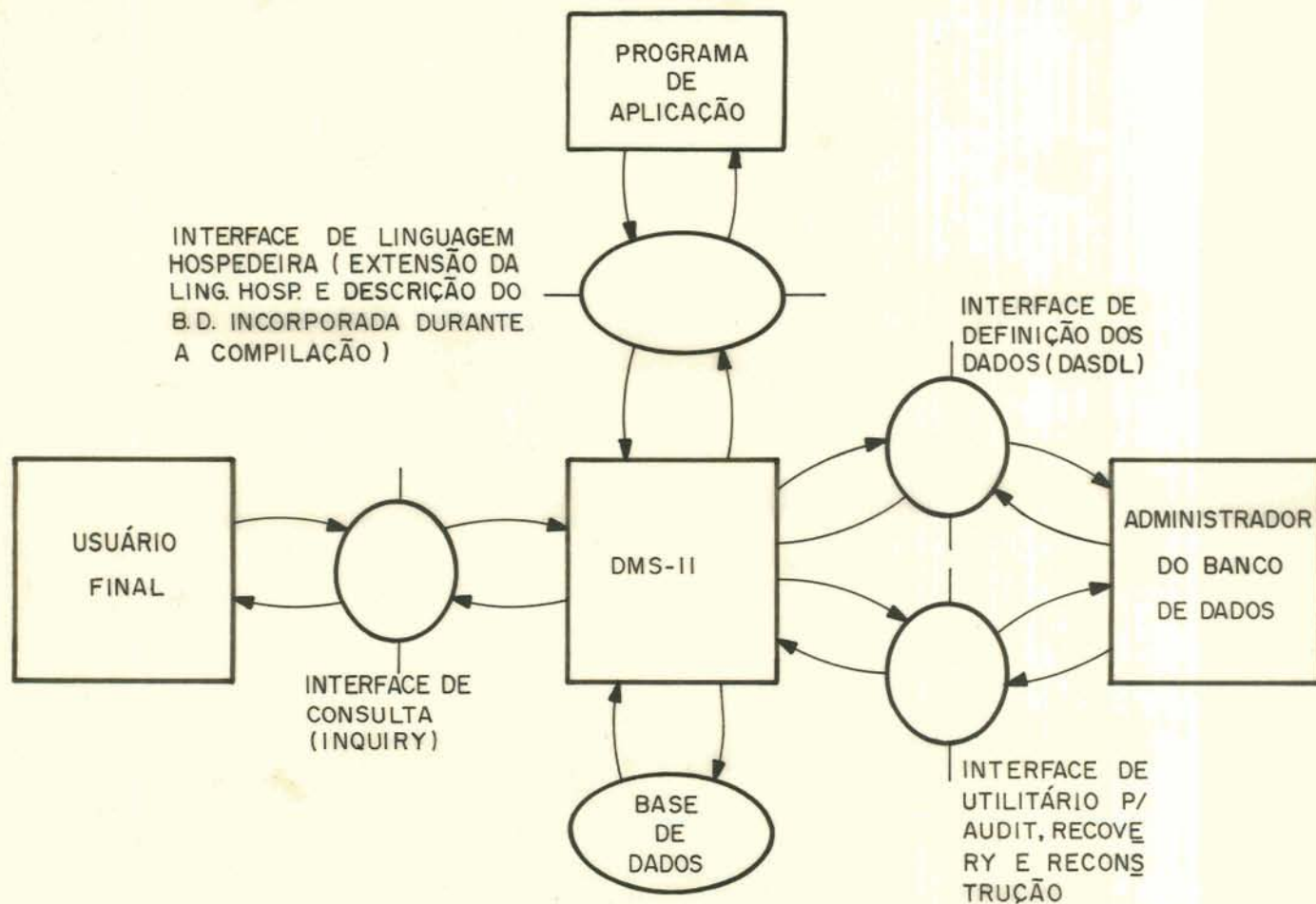


Figura 5.17- Arquitetura funcional do DMS-II

Nesta análise será considerado somente a interface de definição dos dados, sob responsabilidade do administrador do banco de dados e a interface de linguagem hospedeira.

## 5.6.2 Interface de definição dos dados no sistema

### 5.6.2.1 Definição da base de dados

O banco de dados no sistema DMS II é descrito, pelo administrador do banco de dados, através da linguagem DASDL (Data And Structure Definition Language). Esta linguagem oferece comandos que permitem uma descrição detalhada de todos os componentes de um banco de dados, tanto do ponto de vista lógico quanto do ponto de vista físico.

Feita a definição, os comandos fontes, que compõem a descrição, são submetidos ao compilador DASDL que gera um arquivo de descrição do banco de dados (DESCRIPTION/<nome B.D.>) contendo as informações sobre as estruturas descritas e sendo posteriormente usado para a montagem das rotinas de acesso (ACCESS ROUTINES) que permitirão o acesso ao banco de dados durante a execução dos programas do usuário.

A definição dos dados no DMS II é realizada em termos de:

- . Item de dados globais - item que identifica informações sobre o banco de dados como um todo, como por exemplo, ítems do tipo população (POPULATION).

- . Conjunto de dados (DATA-SET) - é semelhante a um arquivo convencional e é formado por uma coleção de registros (RECORDS). Pode conter estruturas embutidas (EMBEDDED) o que possibilita a formação de um esquema hierárquico do tipo mestre-membro.

- . Sets, subsets e access - são estruturas que permitem o acesso e/ou ordenação de um DATA-SET, em uma seqüência lógica especificada. São índices cujas entradas contêm chave de acesso, ponteiro ao DATA-SET e opcionalmente dados, podendo ter diversos SETS, SUBSETS ou ACCESS associados.

dos a um DATA-SET.

Um registro é formado por uma seqüência de itens (campos) sendo que cada item é descrito em termos de tipo de dado e tamanhos. Os tipos podem ser:

- . Alpha (seqüência de caracteres EBCDIC)
- . Boolean (booleano)
- . Number (decimal compactado)
- . Field (campo - valores numéricos ou booleanos)
- . Real (ponto flutuante, com precisão simples).

Na descrição de um registro podem, também, ser incluídos elos ou links (REFERENCE) que são itens destinados a conter endereços de registros pertencentes a outros DATA-SETS. Permitem fazer apontamento entre registros através de endereço simbólico (chave no SET) ou diretamente pelo endereço absoluto em disco. Estes links são mantidos pelo usuário e alguns possuem proteções contra exclusão dos membros apontados. A sintaxe da definição pode ser vista no exemplo abaixo:

```

EMPREGADOS DATA SET
  ( . . . . );
DEPARTAMENTOS DATA SET
  (DEP-NRO      NUMBER(6);
   :
  DEP-CHEFE    REFERENCE TO EMPREGADOS COUNTED;
  );

```

Pode haver a especificação de vários tipos de registros em um mesmo DATA-SET sendo que a identificação é feita por um item de controle que tem associado o número máximo de tipos diferentes permitidos como pode ser visto no exemplo.

```
EMPREGADOS DATA SET
  (TR-CTRL  TYPE(3);
    1:(DADOS-1  ALPHA(256);
      );
    2:(DADOS-2  ALPHA(256);
      );
```

Para permitir a recuperação da integridade do banco de dados em eventuais panes no sistema deverá ser especificado uma opção AUDIT e um comando AUDIT TRAIL para definir as características físicas da área a ser usada para registrar as alterações sofridas pelo banco de dados. Além disso um DATA-SET de reinício (RESTART DATA SET) deve ser definido para armazenar as informações necessárias ao reinício dos programas que sofreram a interrupção.

O administrador do banco de dados pode especificar, para cada DATA-SET, alguns atributos físicos como número de buffers, população esperada, dispositivo de armazenamento, critério de agrupamento físico dos registros, esquema de bloqueio para alteração de registros dos DATA-SETS embutidos, e outros mais. Esta especificação é realizada por alteração dos valores padrões, ou seja, para os atributos que não forem especificados serão assumidos os valores padrões.

O DMS II permite, também, que um determinado DATA-SET seja particionado em diversos arquivos físicos diferentes através de chaves de sets que identificam cada uma das partições. Isto permite que nem todo o DATA-SET precise estar residente quando da execução de um determinado programa de aplicação. Somente as partições que o programa utiliza precisam estar presentes.

```

DEPARTAMENTOS DATA SET
  (CHAVE-LOCAL  ALPHA(2);
   DEPTOS-POR-LOCAL DATA SET
     ( . . . . );
   PARTITION ON DEPTO-SET;
   LOCAL-SET SET OF DEPTOS POR LOCAL
   KEY IS DEPTO-NRO
   PARTITION ON DEPTO-SET;
  );
DEPTO-SET SET OF DEPARTAMENTOS KEY IS CHAVE-LOCAL.

```

Na definição dos DATA-SETS e SETS é possível ter os seguintes tipos de representação física:

. Para os DATA-SETS: direct, random, compact, restart, unordered, além do tipo padrão que é assumido quando não indicado.

. Para os SETS e SUBSETS: bit vector, index random, index sequencial, ordered list, unordered list.

Pode-se ressaltar, portanto, que a sintaxe de definição dos dados é muito semelhante à sintaxe da linguagem COBOL.

O banco de dados de exemplo seria definido usando os seguintes comandos fonte na linguagem DASDL:

```

EMPREGADOS DIRECT DATA SET
  ( EMP-NRO          NUMBER(6)  REQUIRED;
    EMP-NOME         ALPHA (30)  REQUIRED;
    EMP-ENDER        ALPHA (30)  NULL IS BLANKS;
    EMP-IDADE        NUMBER(2)   INITIAL VALUE IS Ø;
    EMP-DATAADM      GROUP
      (EMP-DIA       NUMBER(2) ;
        EMP-MES      NUMBER(2) ;
        EMP-ANO      NUMBER(2)
      );
    EMP-DEPTOS      SUBSET OF DEPARTAMENTOS
    KEY IS DEP-NRO
    NO DUPLICATES ORDERED LIST
  );
EMP-ACCESS ACCESS TO EMPREGADOS KEY IS EMP-NRO;
DEPARTAMENTOS DIRECT DATA SET
  ( DEP-NRO          NUMBER(6) ;
    DEP-NOME         ALPHA (30) ;
    DEP-LOCAL        ALPHA (30) ;
    DEP-EMPS         SUBSET OF EMPREGADOS
      KEY IS EMP-NRO
      NO DUPLICATES ORDERED LIST
  );
DEP-ACCESS ACCESS TO DEPARTAMENTOS KEY IS DEP-NRO;
VENCMES DATA SET
  ( VENC-ANO         NUMBER(2) ;
    VENC-MES         NUMBER(2) ;
    VENCEMP         DATA SET
      (VENC-EMP      NUMBER(6) ;
        VENC-TAB     NUMBER(6) ;
        VENC-VALOR   NUMBER(11,2)
      )
  )
PARTITION ON VENCMES-SET
OPEN PARTITION = 13;

```

```

VENCEMP-SET SET OF VENCEMP
  KEY IS VENC-EMP
  PARTITION ON VENCMES-SET
); OPEN PARTITION = 13;
VENCMES-SET SET OF VENCMES KEY IS (VENC-ANO,VENC-MES);

```

#### 5.6.2.2 Definição de esquemas externos

A definição dos esquemas externos para as aplicações é, também, tarefa do administrador do banco de dados e é realizada no próprio fonte submetido ao compilador DASDL para geração da descrição do banco de dados.

A especificação é realizada em 2 etapas complementares durante a definição dos dados.

A primeira etapa é a especificação a nível de itens que farão parte do registro fornecido para um determinado esquema externo. Esta especificação é realizada com o comando REMAPS que redefinirá o registro com uma máscara lógica igual ou diferente do que consta no layout físico. Defina o registro visto de um ângulo diferente. O exemplo que segue mostra a sintaxe da especificação:

```

EMPREGADOS-RED REMAPS EMPREGADOS
  READONLY ALL GIVING EXCEPTION
  (EMP-NRO,
   EMP-ENDER,
   EMP-NOME-RED = EMP-NOME);

```

A segunda etapa é a especificação do banco de dados lógico a nível de registros que compõem o esquema externo. Neste ponto serão indicadas as estruturas que farão parte deste esquema externo. Opcionalmente, ela pode englobar a primeira etapa de especificação, ou seja, a primeira etapa pode ser omitida e toda a especificação ser realizada através da segunda etapa. A definição do banco de dados lógico



co é feita com o comando DATABASE, como pode ser visto no exemplo.

```
BDLOGICO DATABASE
    (EMPREGADOS-RED,
     EMPREGADOS (EMP-IDADE-RED = EMP-IDADE) ,
     DEPARTAMENTOS,
     DEPARTAMENTOS (SET DEPTO-SET-RED = DEPTO-SET)
    );
```

### 5.6.3 Aspectos funcionais do embutimento

#### 5.6.3.1 Comunicação do usuário com o banco de dados

O usuário visualizará, em seu programa de aplicação, a área de comunicação segundo os 3 canais de comunicação que aparecem na figura 5.18.

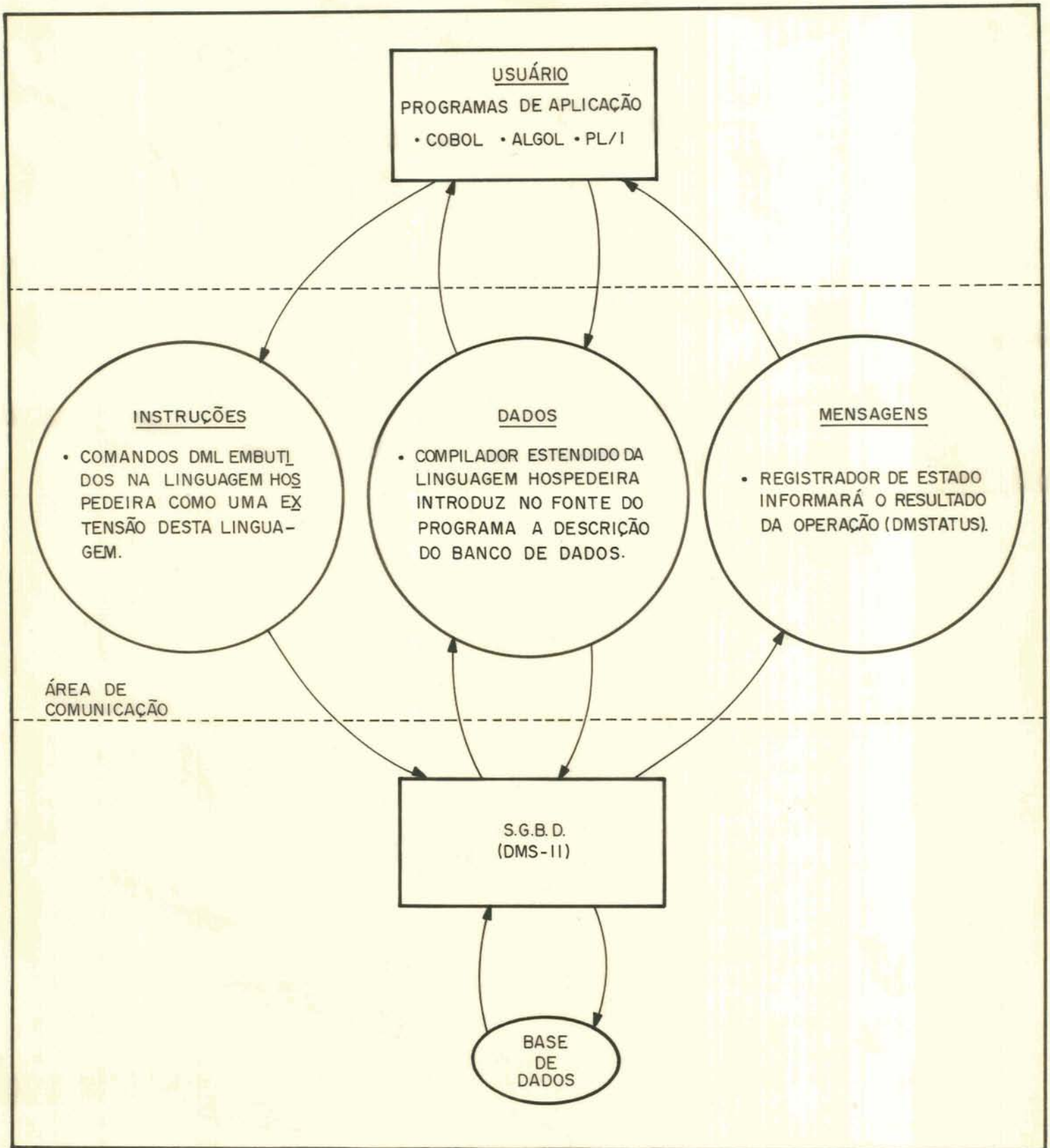


Figura 5.18 - Canais de comunicação programas de aplicação e DMS-11

## a) DADOS

A cada DATA-SET do banco de dados é associada, em cada programa, uma área de trabalho onde os registros são acessados, montados e modificados pelo programa. Nestas áreas as rotinas de acesso colocam os registros recuperados e delas retiram os registros que serão inseridos no banco de dados.

O programador declara, na "data-base section", parte integrante da "data division" do programa, com o uso do comando DB, um banco de dados físico ou lógico indicando as estruturas que o seu programa utilizará.

Ex.:

```
IDENTIFICATION DIVISION.
:
DATA DIVISION.
:
DATA-BASE SECTION.
DB PESSOAL ALL.
Ø1 EMPREGADOS.
Ø1 DEPARTAMENTOS.
:
```

Com esta informação o compilador estendido da linguagem hospedeira introduz, logo após o comando DB, a descrição das estruturas e ítems que compõem o banco de dados referenciado. A descrição é extraída de um arquivo denominado "DESCRIPTION/<nome BD>".

Poderão ser selecionadas todas as estruturas do bando de dados (DB <nome BD> ALL) ou somente as estruturas indicadas (referenciadas pelo nome do DATA-SET).

Opcionalmente, o usuário pode declarar várias á-

reas de trabalho e caminhos associados a mesma estrutura física como pode ser visto no exemplo abaixo.

```

:
DATA-BASE SECTION.
DB PESSOAL.
Ø1 EMP1 = EMPREGADOS USING AUXSET1 = EMP-ACCESS.
Ø1 EMP2 = EMPREGADOS USING AUXSET2 = EMP-ACCESS.

```

#### b) INSTRUÇÕES

Diversos comandos específicos para a manipulação de dados, mantidos em bancos de dados gerenciados pelo DMS II, são oferecidos na extensão de uma das linguagens hospedeiras. Os comandos DML são portanto embutidos na linguagem hospedeira como sendo uma extensão desta linguagem, possuindo uma sintaxe e concepção de uso semelhantes.

Os comandos DML embutidos na linguagem hospedeira permitem a efetivação de consultas e atualizações sobre os bancos de dados anteriormente definidos pelo administrador do banco de dados.

Para ilustrar, segue um exemplo de comando DML.

```

:
PROCEDURE DIVISION.
:
FIND EMP-ACCESS AT EMP-NRO = AUX-EMP.
:

```

Com respeito ao canal de instruções e o ambiente que envolve o seu correto manuseio, convém destacar alguns conceitos importantes. Estes conceitos dizem respeito ao processo de navegação sobre as estruturas do banco de dados e são os seguintes:

. Caminho corrente (CURRENT PATH) - se refere a uma posição de um DATA-SET, SET ou SUBSET. Cada estrutura possui um ou mais cursores do tipo "CURRENT PATH" associados e pode apontar para um registro válido (neste caso considerado "DEFINED") ou para uma posição sem registro (considerado "UNDEFINED").

. Registro corrente (CURRENT RECORD) - é o registro do DATA-SET apontado pelo cursor do "CURRENT PATH".

Conhecendo estes conceitos e a sintaxe dos comandos o programador estará apto a codificar as instruções DML que permitirão o acesso aos dados armazenados no banco de dados. Deverá também, com o auxílio de comandos da linguagem hospedeira, montar os procedimentos que controlem o fluxo de informações dentro do programa.

Complementando a análise dos aspectos vinculados às instruções DML no ambiente da linguagem hospedeiras um quadro das funções dos principais comandos do DMS II é apresentado na figura 5.19.

CONTROLE	COMANDO	FUNÇÃO
CONTROLE	OPEN	- ABERTURA DO BANCO DE DADOS
	CLOSE	- FECHAMENTO DO BANCO DE DADOS
	BEGIN-TRANSACTION	- INÍCIO DE ESTADO DE TRANSAÇÃO PARA ALTERAÇÕES
	END-TRANSACTION	- FIM DE ESTADO DE TRANSAÇÃO
	IF	- TESTE SE O CONTEÚDO DE UM ITEM DE DADO TEM VALOR NULO OU TESTE DO CÓDIGO DE ESTADO
	DMTERMINATE	- FINALIZAR O PROGRAMA DE FORMA ANORMAL
RECUPERAÇÃO	FIND	- LOCALIZA E RECUPERA REGISTROS CONFORME O CRITÉRIO DE SELEÇÃO. EXISTEM VARIAS MODALIDADES PARA ESTE COMANDO. POR EXEMPLO:  <ul style="list-style-type: none"> <li>◦ FIND &lt;NOME - B. D.&gt; ( DADOS GLOBAIS )</li> <li>◦ FIND &lt;NOME DATASET&gt; AT ( ACESSO DIRETO )</li> <li>◦ FIND NEXT ( ACESSO SEQUENCIAL )</li> <li>◦ FIND KEY OF ( ACESSO A CHAVE )</li> </ul>
	LOCK / MODIFY	- IDEM COMANDO "FIND" COM CHAVEAMENTO
MODIFICAÇÃO	CREATE	- INICIALIZA "RECORD AREA" DO DATA-SET
	STORE	- TRANSFERE A IMAGEM DA "RECORD AREA" PARA O BANCO DE DADOS
	ASSIGN	- ESTABELECE UM LINK ENTRE REGISTROS
	INSERT	- INSERÇÃO DE ENTRADA EM SUBSET
	REMOVE	- REMOÇÃO DE ENTRADA EM SUBSET
	SET	- ATRIBUIÇÃO DE POSIÇÃO DE ACESSO A UM SET OU ATRIBUIÇÃO DE VALOR NULO A UM ITEM DE DADO
	DELETE	- REMOÇÃO DE REGISTRO DE UM DATA-SET
	FREE	- LIBERA BLOQUEIO DE REGISTRO
	GENERATE	- CRIAÇÃO DE TODO UM SUBSET MANUAL A PARTIR DE UM OU DOIS OUTROS SUBSET'S
COMPUTE	- ATRIBUIÇÃO DE VALOR A UM ITEM BOOLEANO	

Figura 5.19 : Quadro de instruções DML do DMS - II

## c) MENSAGENS

Na execução da maioria dos comandos de manipulação de dados pode, ocasionalmente, resultar uma condição de exceção que deverá ser tratada por alguma rotina do programa de aplicação. Algumas condições retornadas podem consistir em erros enquanto que outras exigem, somente, uma intervenção que controle um fluxo de dados adverso.

As exceções e suas origens podem ser detectadas no programa de aplicação verificando o conteúdo de um registrador denominado "DMSTATUS".

Ex.: IF DMSTATUS(<tipo de exceção>)

Como tipo de exceção pode ser considerado, por exemplo, NOTFOUND (registro não localizado), NORECORD (caminho corrente inválido), DUPLICATES (chave duplicada), NOT-LOCKED (registro não bloqueado) e outros mais.

O registrador "DMSTATUS" normalmente é usado em combinação com uma cláusula "ON EXCEPTION" ou "USE ON DMERROR" para estruturação de um procedimento de tratamento de exceções no programa.

Ex.:

```

:
PROCEDURE DIVISION.
DECLARATIVES.
TRATA-EXCEÇÃO SECTION.
    USE ON DMERROR.
VERIFICA-EXCEÇÃO.
:
    IF DMSTATUS (NOTFOUND)
:
END DECLARATIVES.
:

```

#### 5.6.3.2 Conversões

O tipo de conversão que pode ser constatada no sistema DMS II seria o da adaptação da representação dos dados para os padrões da linguagem hospedeira, pois o usuário recebe, inserido em seu programa, a descrição completa dos dados numa representação que é reconhecível pelo compilador desta linguagem hospedeira. Não existe, no entanto, a possibilidade do usuário especificar um formato diferente do que vem especificado nesta descrição introduzida no fonte do programa.

#### 5.6.3.3 Estruturas de controle de fluxo

A estrutura de controle de iteração é estabelecida através do uso de comandos da linguagem hospedeira e do teste explícito do conteúdo do registrador de estado da operação (DMSTATUS).

#### 5.6.3.4 Considerações sobre proteção dos dados

Para garantir aspectos de segurança ao sistema, o DMS II dispõe do mecanismo de definição de esquemas externos e especificação de banco de dados lógicos que auxiliam no sentido de restringir o acesso aos dados armazenados. Com o uso desses esquemas externos o usuário manipulará somente os dados que são pertinentes ao seu programa de aplicação, não tendo acesso aos demais.

Com relação à manutenção da integridade dos dados, o sistema põe à disposição do responsável pela definição das estruturas algumas facilidades como é o caso da cláusula "REQUIRED" que assegura que o item de dado (campo) que tiver esta cláusula associada, necessariamente deverá estar preenchido quando da inclusão do registro, ou ainda a



cláusula "VERIFY" que assegura que condições prévias sejam satisfeitas antes do armazenamento do registro, ou então uma cláusula "NO DUPLICATES" que proíbe o armazenamento de registros duplicados. Possui ainda facilidades que permitem indicar que um determinado item de dado ou registro somente pode ser lido (READONLY) e não modificado.

Para a proteção do sistema, o DMS II implementa um mecanismo de "LOGGING & RECOVERY" que permite a recuperação do banco de dados após uma falha de programa ou pane no sistema. Este mecanismo permite retroceder as alterações feitas até um determinado ponto especificado (ROLLBACK) e realizar a recuperação até um instante em que as informações armazenadas estão íntegras.

O administrador do banco de dados tem também a sua disposição, quando da definição das estruturas físicas e lógicas, facilidades que permitem forçar um bloqueio de registros na alteração destes em DATA-SETS embutidos (LOCK TO MODIFY DETAILS).

Além disso existe, também, um mecanismo para chaveamento de informações num ambiente de atualização concorrente (comando LOCK para registros).

#### 5.6.3.5 Considerações sobre homogeneidade

Existe no DMS II uma semelhança de sintaxe visível com a linguagem COBOL. A mesma regra de formação de identificadores e a mesma forma de estruturação dos registros pode ser observada. Também a sintaxe dos comandos DML se aproximam muito à sintaxe dos comandos normais da linguagem COBOL.

Quanto a homogeneidade de sintaxe entre o DMS II e as outras linguagens hospedeiras (ALGOL, PL/1), pode-se considerar que as extensões destas linguagens procuram se-

guir um certo padrão de semelhança com a extensão do COBOL, exceto por pequenas diferenças nas declarações e manipulação de itens particulares. Existe, no entanto, uma certa diferenciação conforme os padrões sintáticos de cada linguagem apesar de utilizar os mesmos mnemônicos em todas linguagens hospedeiras, para denominar os comandos DML.

#### 5.6.4 Aspectos de implementação do embutimento

##### 5.6.4.1 Descrição da forma de embutimento

O DMS II implementa o embutimento da linguagem de manipulação de dados nas linguagens hospedeiras por intermé dio de uma adaptação desta linguagem hospedeira a fim de que aloje o conjunto de comandos DML que proporcionarão a recuperação e atualização dos dados armazenados no banco de dados. O compilador da linguagem hospedeira, portanto, é mo dificado para que admita novas construções relativas a de- claração dos dados e comandos específicos de manipulação de dados que serão adicionadas ao conjunto atual de construções obedecendo, é claro, à sintaxe usada no contexto origi- nal da linguagem.

Baseado nisto, o processo de interface do progra- ma de aplicação com o sistema DMS II será constituído das seguintes etapas:

. Inicialmente o administrador do banco de dados (ABD) realiza a definição das estruturas, submetendo o fon- te ao compilador DASDL que irá gerar um arquivo (DESCRIPTION) contendo informações sobre as estruturas descritas.

. De posse do arquivo "DESCRIPTION" o ABD subme- te-o, juntamente com os simbólicos de geração do software do banco de dados, a um compilador DMALGOL que produzirá as rotinas de acesso (ACCESS ROUTINES) utilizadas em tempo de execução pelos programas de aplicação, além de outros arqui

vos destinados à recuperação de dados em caso de pane no sistema.

. Iniciando a etapa que fica sob responsabilidade do programador de aplicação, os programas fontes são submetidos ao compilador estendido da linguagem hospedeira que, utilizando o arquivo "DESCRIPTION" por intermédio de um módulo denominado "DATABASE/INTERFACE", introduzirá no fonte do programa uma descrição das estruturas especificadas pelo programador e produzirá um código objeto executável.

. De posse do código objeto executável, o programador poderá ativá-lo para que este, com auxílio das rotinas de acesso (ACCESS ROUTINES), possa realizar a recuperação ou atualização de dados armazenados no banco de dados.

O processo completo pode ser visto na figura 5.20.

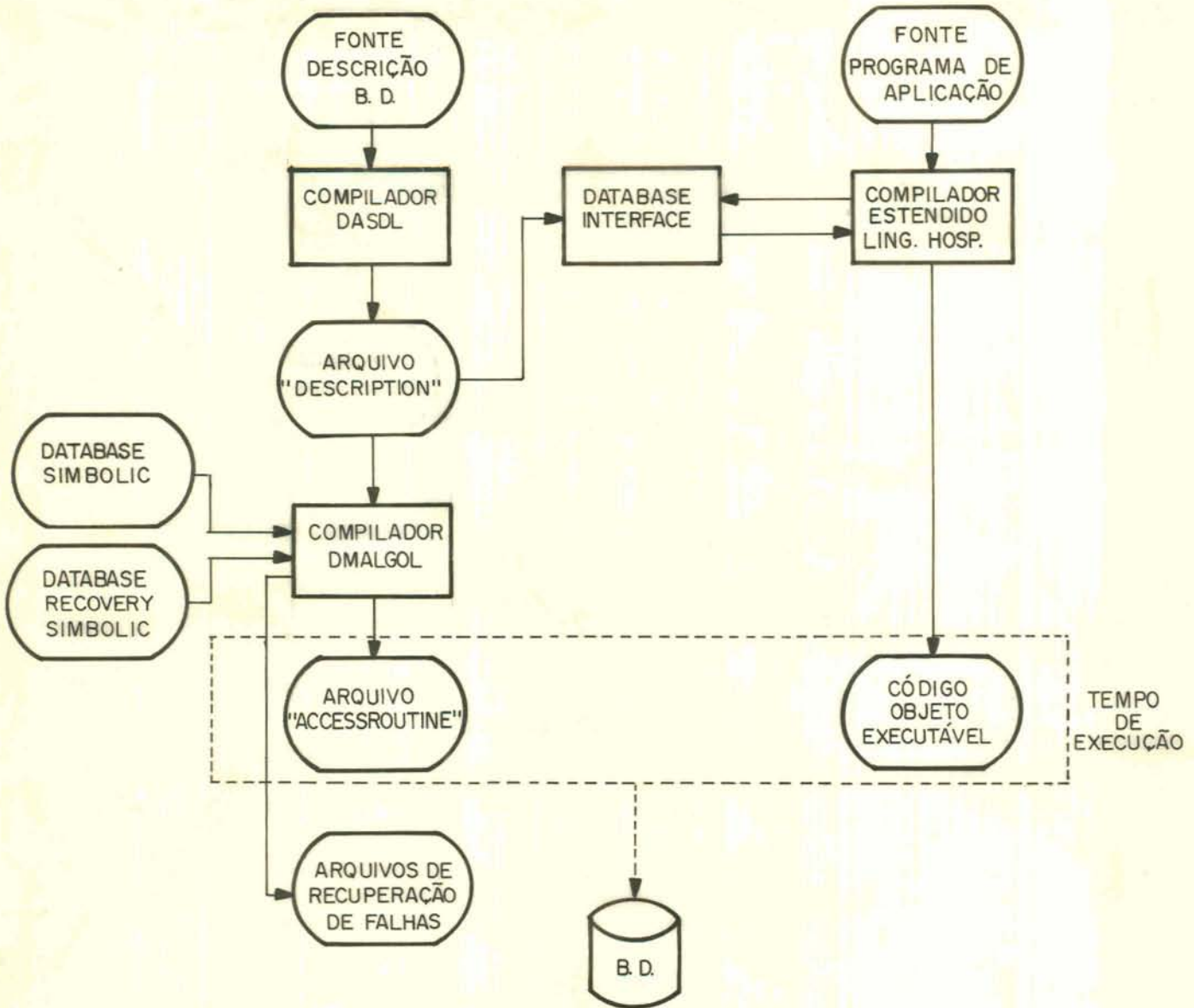


Figura 5.20 - O DMS-II embutido em linguagem hospedeira

#### 5.6.4.2 Considerações sobre a portabilidade da linguagem

A tarefa de implementação da forma de embutimento adotada pelo DMS II torna-se muito onerosa em vista de que requer uma inspeção completa e reescrita considerável de rotinas nos compiladores das linguagens hospedeiras, além de necessitar de grandes esforços para uma padronização entre as diversas linguagens.

Vê-se, portanto, que o DMS II não é muito portátil.

## 5.7 O Sistema INGRES

### 5.7.1 Caracterização do sistema

#### 5.7.1.1 Introdução

O INGRES (Interactive Graphics and Retrieval System) é um S.G.B.D. [/ALL 76/, /HEL 75/, /STO 76/, /STO 80/] desenvolvido pelo BELL TELEPHONE LABORATORIES e implementado no sistema operacional UNIX para computadores PDP 11/40, 11/45 e 11/70.

Como partes operacionais do INGRES são oferecidas as linguagens QUEL e EQUCEL. O QUEL (Query Language) é uma sublinguagem de dados relacional que proporciona as facilidades de definição, manipulação e consulta de dados armazenados no sistema INGRES. O EQUCEL (Embedded QUEL), por sua vez, é resultante do embutimento da linguagem QUEL na linguagem de programação C.

#### 5.7.1.2 Estruturas de dados

O sistema INGRES utiliza a abordagem relacional onde os dados armazenados e gerenciados por este sistema são estruturados através de uma coleção de relações em uma representação tabular sendo que cada coluna da tabela corresponde a um domínio da relação.

O banco de dados INGRES é considerado uma coleção de relações, cada qual identificada através de um nome a ela associado, e que são organizadas e mantidas por um administrador de banco de dados.

### 5.7.1.3 Linguagens hospedeiras

A linguagem EQU~~EL~~ foi projetada para permitir o embutimento do QUEL na linguagem hospedeira C que é uma linguagem de alto nível na qual o próprio sistema operacional UNIX do PDP 11 está escrito.

Para o projeto da linguagem EQU~~EL~~, algumas metas foram impostas e dentre as quais podem ser citadas:

- . A nova linguagem deve ter a capacidade integral da linguagem hospedeira C e da linguagem de banco de dados QUEL.

- . O programa escrito na linguagem C deverá ter a capacidade de processar individualmente cada tupla que satisfizer a qualificação obtida pelo comando de recuperação da linguagem QUEL.

A linguagem EQU~~EL~~ tem pontos em comum com a linguagem SQL ou ainda outras linguagens como ALPHA e SQUARE na forma de que se trata de uma linguagem que libera o programador da preocupação de como as estruturas de dados são implementadas e quais os algoritmos que operam no armazenamento destes dados. Estas facilidades dão à linguagem um alto grau de independência de dados.

### 5.7.1.4 Estrutura funcional

A figura 5.21 apresenta a arquitetura funcional do sistema INGRES.

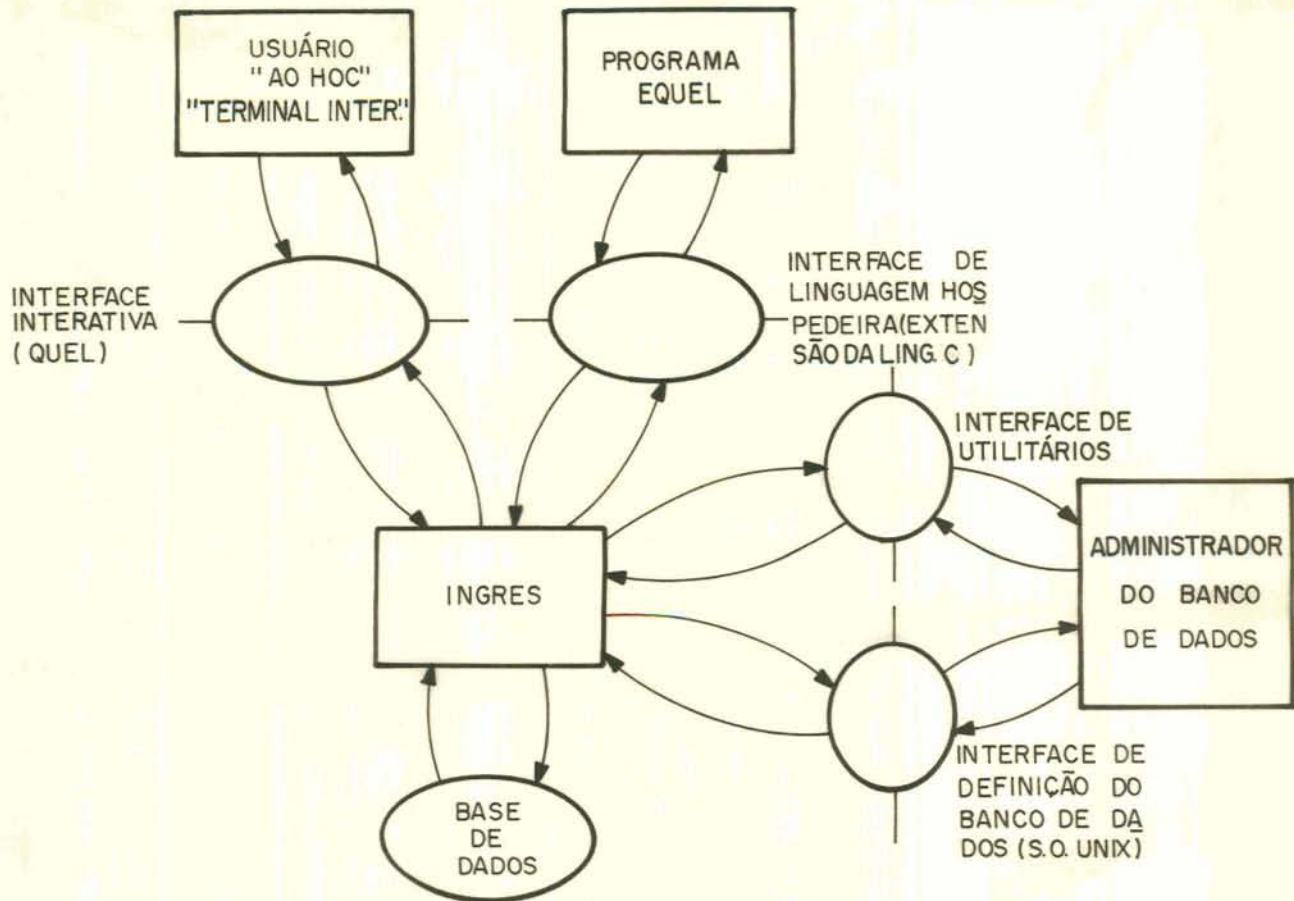


Figura 5.21 - Arquitetura funcional do INGRES

## 5.7.2 Interface de definição dos dados no sistema

### 5.7.2.1 Definição da base de dados

Antes da utilização do banco de dados pelos programas de aplicação ou/e pelo monitor de terminal interativo, este deve ser inicializado e suas relações geradas. Isto é tarefa do administrador do banco de dados que o fará através da interface com o sistema operacional UNIX.

A definição da base de dados é realizada por uma



série de comandos utilitários que permitem a definição de relações e de restrições de integridade sobre estas, além da especificação da estrutura de armazenamento e de índices para as relações.

Os comandos utilizados são os que seguem:

INGRES <nome-do-BD> : Abre a comunicação do usuário com o INGRES.

CREATEDB <nome-do-BD> : Inicializa um banco de dados.

DESTROYDB <nome-do-BD> : Elimina um banco de dados.

CREATE <nome-relação> (<nome-dominio> IS <formato, <nome-dominio> IS <formato, ...>)  
: Define uma relação.

DESTROY <nome-relação> : Elimina uma relação.

INTEGRITY CONSTRAINT <condição>  
: Define restrições de integridade.

MODIFY <nome-relação> TO <estrutura-armaz.> ON(<chave 1>, <chave 2>...)  
: Altera a estrutura de armazenamento.

INDEX ON <nome-relação> IS <nome-do-Índice> (<chave 1>, <chave 2>, ...)  
: Cria índices.

De um modo geral, portanto, a definição da base de dados, criação e eliminação de relações, atribuição e revogação de asserções de integridade ficam a cargo do administrador do banco de dados que também é responsável pela modificação do método de acesso e inclusão de índices.

Entretanto, existe a possibilidade de outros usuários (que não o administrador do banco de dados) definirem relações e, portanto, alterarem o esquema definido, permitindo esquemas dinâmicos [/ALL 76/]. Através dos comandos RETRIEVE INTO e CREATE o usuário cria novas relações que podem ser tornadas permanentes (através do comando SAVE) ou que serão mantidas por um período de tempo especificado pelo administrador do banco de dados. No INGRES, relações cria

das durante a execução não desaparecem quando o programa termina.

#### 5.7.2.2 Definição de esquemas externos

O INGRES propicia, também, a facilidade de definição de esquemas externos, chamadas vistas ("views").

Estas são definidas através do comando DEFINE cuja sintaxe é extremamente semelhante ao comando RETRIEVE.

Ex.:

```
## ## RANGE OF E IS EMP
## ## DEFINE EMPANT(E.NOME, E.NRODEPT, E.SALARIO,
## ##                E.GERENTE, E.IDADE)
## ##                WHERE E.IDADE < 30
```

São consideradas vistas válidas aquelas que podem ser materializadas através de uma consulta pelo comando RETRIEVE.

As vistas permitem ao INGRES aceitar programas EQUERL escritos sobre versões obsoletas do banco de dados e também para assegurar que determinados dados sejam inacessíveis por determinados usuários não qualificados.

Trata-se de relações virtuais que não existem fisicamente.

Os predicados de definição das vistas são mantidos pelo INGRES nos catálogos do sistema, em relações específicas para esta finalidade ("VIEW CATALOG").

Quando do uso de uma determinada vista por um programa de aplicação escrito na linguagem EQUERL o precompilador realiza uma pesquisa nos catálogos do sistema, extrai o predicado de definição da vista, modificando os comandos

fontes para o texto que havia sido definido como vista para este usuário.

Supondo que o programa de aplicação tenha, em seu código fonte, os seguintes comandos:

```
## ## RANGE OF L IS EMPANT.  
## ## REPLACE L (SALARIO = 9*L.SALARIO)  
## ## WHERE L.NOME = "PEDRO"
```

Utilizando a vista definida anteriormente, o pre-compilador modificará os comandos fontes para:

```
RANGE OF E IS EMP  
REPLACE E(SALARIO = 9*E.SALARIO)  
WHERE E.NOME = "PEDRO"  
AND E.IDADE < 30
```

### 5.7.3 Aspectos funcionais do embutimento

#### 5.7.3.1 Comunicação do usuário com o banco de dados

A área de comunicação do usuário com o sistema de gerência de banco de dados INGRES aparece esboçada na figura 5.22.

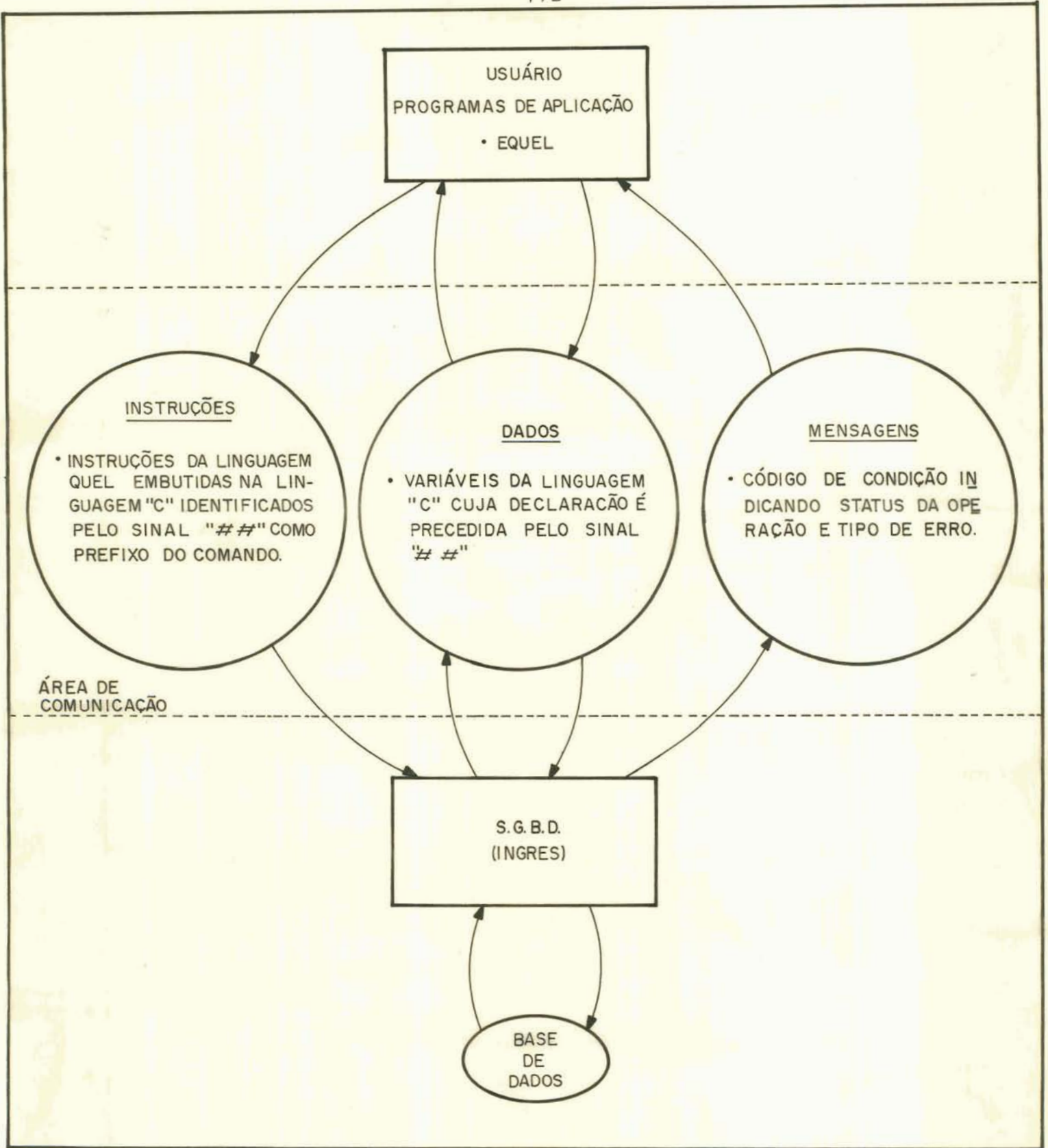


Figura 5.22- Canais de comunicação programas de aplicação e INGRES

## a) DADOS

Qualquer declaração da linguagem QUEL é válida como declaração EQUQL desde que esta venha precedida pelo sinal "## #".

As variáveis do programa C podem ser usadas nos comandos QUEL no lugar dos nomes de relações, nomes dos domínios, elementos de argumento ou valores dos domínios. Para tal estas variáveis também deverão ser declaradas da mesma forma, ou seja, precedidas pelo duplo sinal cardinal (## #).

O exemplo, a seguir, ilustra o uso das variáveis em um programa EQUQL.

```
## # CHAR  NOMEAUX [30];
## # INT   SALARAUX;
## # CHAR  GERENTEAUX [30];
PROG ( );
{
## # INGRES PESSOALDB /*INVOCA O INGRES C/B.D. APROPRIADO*/
## # RANGE OF E IS EMP
## # RETRIEVE (NOMEAUX = E.NOME, SALARAUX = E.SALARIO,
## #          GERENTEAUX = E.GERENTE)
## # WHERE E.NRODEPT = 2440
## # {
      < PROCESSA REGISTRO >
## # }
```

Os valores dos campos no registro recuperado (nome, salário, gerente) são copiados para as variáveis do programa (nomeaux,salaraux, gerenteaux) antes de cada iteração.

Nome de relações e domínios especificados quando da criação das relações, pelos comandos utilitários de definição, podem ser utilizados naturalmente no programa sem a

necessidade de declará-los, pois o precompilador se encarregará de extrair as informações sobre estes objetos do catálogo do sistema.

#### b) INSTRUÇÕES

Uma vez inicializado pelo administrador do banco de dados através de comandos utilitários do INGRES ou do sistema operacional UNIX, este poderá ser manipulado por instruções dos programas de aplicação escritos na linguagem EQUERL.

Na linguagem C os comandos EQUERL são identificados através do sinal "# #", anteposto ao referido comando.

Ex.:

```
# # RETRIEVE (SAL = X.SALARY)
# # WHERE X.NAME = EMPNAME
# # {
      <código da linguagem C>
# # }
```

No INGRES existe o conceito de variáveis do tipo "tupla" (tuple variables) que são usadas como argumento para tuplas. Estas variáveis são especificadas através da declaração "RANGE", na forma como é apresentado abaixo:

```
RANGE OF <lista-de-variáveis> IS <nome-da-relação>
```

Esta declaração tem o propósito de especificar a relação sobre a qual cada variável se limita e deverá ser especificado antes de efetuar qualquer operação de manipulação de dados sobre a relação em questão.

As variáveis "tupla" são parte integrante do S. G.B.D. e não do canal de comunicação (área de dados) no programa de aplicação, motivo pelo qual não foram incluídas na

descrição do canal de dados.

Convém ressaltar que a atuação das instruções é sobre um conjunto de tuplas que são recuperadas simultaneamente, orientadas pela qualificação feita na seleção.

A linguagem EQUQL admite o uso direto de expressões aritméticas nos comandos sem necessitar, para isto, intervenção da linguagem hospedeira.

Ex.: Aumentar o salário de João da Silva em 10%.

```
REPLACE E(SALARIO BY 1.1*E.SALARIO)
WHERE E.NOME = 'JOAO DA SILVA'
```

A linguagem QUEL dispõe, também, de algumas funções que operam sobre um agregado de tuplas (grupos). São elas:

- . AVG, AVG' - valor médio
- . SUM, SUM' - somatório
- . COUNT, COUNT' - contador de ocorrências
- . MAX - valor máximo
- . MIN - valor mínimo

OBS.: O apóstrofo seguindo a função indica que valores duplicados não serão considerados.

Ex.: Atribuir o salário médio dos funcionários do departamento 2440 a todos os funcionários do departamento 2441.

```
RANGE OF E IS EMP.
REPLACE E(SALARIO BY
          AVG'(E.SALARIO WHERE E.NRODEPT = '2440')
WHERE E.DEPT = '2441'
```

De uma forma geral, pode-se considerar que os comandos QUEL seguem o seguinte padrão:

```

<comando> [<nome-resultante>] (<target-list>)
      [WHERE <qualificação>]
<comando> ::= RETRIEVE | APPEND | REPLACE | DELETE
              ↓           ↓           ↓           ↓
              Recupera  Inclu-  Modifi-  Exclu-
              ção      são      cação    são

<nome-resultante> ::= "nome da relação que qualifica
                      as tuplas (para RETRIEVE e
                      APPEND" | "nome de uma variável
                      tupla que através da quali-
                      ficação identifica a tupla (pa-
                      ra REPLACE e DELETE)"

<target-list> ::= "LISTA NA FORMA: <domínio-da-resul>=
                  FUNÇÃO ..."

```

### c) MENSAGENS

Mensagens de erro são passadas pelos condutos que interligam cada processo do sistema e são retornados ao usuário através de um código de condição, possibilitando o seu teste e tratamento adequado.

Além disso, o INGRES provê, também, facilidades para manipulação de exceções controladas por rotinas padrões do sistema (procedure II error) ou rotinas definidas pelo próprio usuário [/ALL 76/, /STO 77/].

### 5.7.3.2 Conversões

Na atual implementação do INGRES, tipos de dados complexos não são manipulados, estando definidos somente aqueles tipos básicos que comumente são encontrados nas linguagens convencionais, ou seja, inteiros, reais, booleano, array's, etc. O mesmo acontece com a linguagem C.

Já que o precompilador não pode, de antemão, conhecer qual tipo de dado os vários domínios terão durante a execução, os projetistas do sistema INGRES tinham, como al-



ternativa, para sanar este problema:

a) Insistir que o tipo de uma variável que é usada no programa C coincida com o tipo do valor retornado do INGRES. Esta solução, porém, não pode ser aceita pois vai contra os princípios de independência de dados.

b) Executar, em tempo de execução, a conversão entre todos os tipos de variáveis da linguagem C e todos os tipos de domínios do INGRES.

A solução adotada foi a de executar [/ALL 76/, STO 77/], em tempo de execução, as conversões necessárias, tarefa esta facilitada em virtude do limitado acervo de tipos de dados manipulados pelo INGRES e pela linguagem C.

### 5.7.3.3 Estruturas de controle de fluxo

No INGRES a lógica de iteração para recuperação de tuplas e liberá-las ao programa de aplicação é implicitamente controlado pelo próprio comando da DML. O usuário não necessita controlar explicitamente o fim da iteração. O usuário deverá somente especificar, usando comandos da linguagem hospedeira, os procedimentos que deseja realizar para cada tupla retornada ao programa de aplicação. Estes comandos da linguagem hospedeira são codificados dentro do bloco que compõe a instrução EQUER, como pode ser visto no exemplo abaixo:

```

      ## ## RANGE OF E IS EMP
      ## ## RETRIEVE ...
      ## ## WHERE ...
      ## ## {
      ## ##   <comandos da ling.hosp. dentro do bloco de iteração >
      ## ## }
      ## ## FIM DO BLOCO DE ITERAÇÃO
      ## ## INÍCIO DO BLOCO DE ITERAÇÃO

```

#### 5.7.3.4 Considerações sobre proteção dos dados

Em termos de segurança o sistema INGRES apresenta uma facilidade de permitir que sejam impostas restrições seletivas para o acesso de um determinado usuário às informações contidas no banco de dados.

No INGRES, o fornecimento de autorizações, geralmente, fica sob responsabilidade do administrador do banco de dados.

O privilégio do acesso a determinados dados ou relações do banco de dados é fornecido através do uso do comando RESTRICT ou PROTECT, cuja sintaxe é muito semelhante ao do comando RETRIEVE.

Exemplificando, a atribuição do privilégio ao senhor João da Silva de acessar na relação EMP, suas próprias informações ou as informações de todos os seus subordinados, poderia ser especificada da seguinte forma:

```
RANGE OF E IS EMP
RESTRICT ACCESS FOR 'JOAO DA SILVA' TO EMP
WHERE E.NAME = 'JOAO DA SILVA' OR
      E.GERENTE = 'JOAO DA SILVA'
```

Também vinculado ao aspecto de segurança, o sistema oferece a facilidade de definição de vistas ("views") como já descrito anteriormente.

Para o controle de integridade, o INGRES tem a opção de especificar asserções de integridade. Estas asserções de integridade são especificadas através do comando INTEGRITY CONSTRAINT, geralmente, também, sob responsabilidade do administrador do banco de dados.

O exemplo a seguir mostra o uso deste comando.

```
RANGE OF E IS EMP  
INTEGRITY CONSTRAINT IS E.SALARIO ≤ 21000
```

Estes comandos são armazenados no catálogo do sistema em uma relação conhecida como "INTEGRITY CATALOG" para serem posteriormente usadas pelo precompilador para modificar uma consulta ou atualização.

Este assunto é amplamente discutido na bibliografia [STO 75/] onde também são apresentados alguns algoritmos de implementação destas facilidades.

No que se refere à proteção, o INGRES mantém chaveamento implícito em cada comando de atualização a nível de relação. A informação de chaveamento é mantida em relações específicas para esta função, denominadas "LOCK RELATION". A necessidade de chaveamento total de uma relação deve-se ao fato de não existir um mecanismo de semáforos implementado no sistema operacional UNIX.

#### 5.7.3.5 Considerações sobre homogeneidade

Homogeneidade ou uniformidade de sintaxe é uma das propriedades que podem ser atribuídas à linguagem QUEL e EQUCEL.

A linguagem EQUCEL, utilizada pelos programas de aplicação, está embasada sobre a linguagem QUEL, que é a linguagem que o usuário manipula através do monitor de terminal interativo. Os comandos fontes utilizados para realizar uma consulta através do terminal são sintaticamente idênticos aos utilizados no programa de aplicação.

Convém salientar, no entanto, que, analisando a existência de homogeneidade entre a DML e os comandos da linguagem hospedeira, percebe-se que a DML segue padrões próprios independentes dos padrões da linguagem hospedeira.

#### 5.7.4 Aspectos de implementação do embutimento

##### 5.7.4.1 Descrição da forma de embutimento

Para implementar a linguagem EQUCEL, um tradutor (precompilador) foi escrito a fim de converter um programa EQUCEL em um programa da linguagem C válido, com comandos QUEL convertidos para um apropriado código C e chamadas (CALL's) às rotinas de interface com o banco de dados INGRES.

O programa C resultante é então submetido a um compilador C convencional, produzindo um módulo executável. Além disso, quando um programa EQUCEL é ativado, o módulo executável produzido pelo compilador C é usado como um processo "front end", idêntico ao caso de uma consulta interativa.

Convém esclarecer que no sistema operacional UNIX, onde o INGRES está implementado, o conceito de processo é visto como um determinado programa em estado de execução e que processos podem ser executados independentemente ou podem se comunicar uns com os outros através de um fluxo de dados denominado "PIPES", como pode ser visto na figura 5.23.

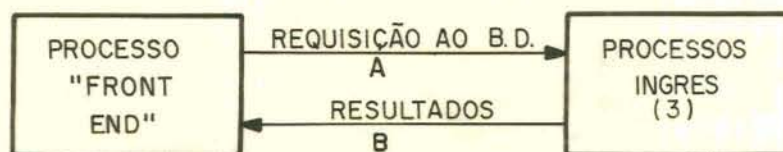


Figura 5.23 - Estrutura do processo INGRES

Durante a execução do programa "front-end" requisições ao banco de dados (comandos QUEL no programa EQUQL) são passadas através do conduto "A" e processadas pelo INGRES. Se tuplas tiverem que ser retornadas durante o processamento, elas serão passadas pelo conduto "B", que também conduzirá um código de condição que indicará sucesso na operação ou o tipo de erro encontrado.

As funções básicas que são desempenhadas pelo precompilador, são as seguintes:

a) Inserir chamadas ao sistema para criar, em tempo de execução, a estrutura do processo.

b) Perceber declarações de variáveis da linguagem C precedidas por duplo sinal cardinal ( $\#\#$ ) como válidas para inclusão nos comandos INGRES.

c) Processar outras linhas fontes precedidas por " $\#\#$ ". Estas linhas são analisadas sintaticamente ("PARSED") para isolar as variáveis da linguagem C. Além disso comandos da linguagem C são inseridos para gerar a requisição, feita pela linha de QUEL, jogando-a no conduto "A" em forma de caracteres modificados para que os valores sejam substituídos por alguma variável da linguagem C.

d) Inserir comandos da linguagem C para ler do conduto "B" a informação resultante e chamar a rotina de tratamento de erro para analisar algum eventual erro retornado no código de condição.

e) Caso dados sejam retornados pelo conduto "B", EQUQL deverá, também:

e1) inserir comandos da linguagem C para ler do conduto "B" uma tupla formatada como um par tipo/valor;

e2) inserir comandos da linguagem C para substituir valores nas variáveis C declaradas

- na lista de argumentos ("target-list") e se necessário realizar as devidas conversões de tipo;
- e3) inserir comandos da linguagem C para passar o controle para o bloco da linguagem C que segue ao comando de consulta QUEL processado;
  - e4) inserir comandos da linguagem C seguindo o bloco mencionado no passo e3 para retornar ao passo e1 caso existirem mais tuplas a serem processadas.

#### 5.7.4.2 Considerações sobre a portabilidade da linguagem

Considerando que o projeto da linguagem EQU~~EL~~ está baseado no embutimento da linguagem de manipulação de dados QUEL na linguagem hospedeira C e que o sistema de gerência de banco de dados INGRES foi proposto e desenvolvido para ter o suporte do sistema operacional UNIX, pode-se afirmar que, para embutir a linguagem QUEL em outra linguagem hospedeira, os quesitos indispensáveis seriam:

- . Compatibilidade da linguagem hospedeira com a estrutura do sistema operacional UNIX (tratamento dos processos).

- . A nova linguagem hospedeira possuir, em sua estrutura, facilidades para chamadas externas a subrotinas.

- . Compatibilidade com os tipos de dados do INGRES (problema de conversões).

- . Projeto e desenvolvimento de um novo precompilador específico para a nova linguagem hospedeira.

## 5.8 O Sistema R

### 5.8.1 Caracterização do sistema

#### 5.8.1.1 Introdução

O SISTEMA R é um S.G.B.D. [/AST 76/, /CHA 75/, /CHA 76/, /CHA 80/, /CHA 81/, /IBM 82a/, /IBM 82b/] que proporciona uma interface de dados relacional de alto nível. É um sistema experimental projetado e desenvolvido no período 1974 a 1975 pela IBM San Jose Research Laboratory.

O sistema sustenta a sublinguagem de dados SQL (originalmente conhecida como SEQUEL) como uma linguagem de consulta para acesso interativo ou então usada em interface com linguagens hospedeiras.

Atualmente a linguagem SQL já ultrapassou as fronteiras de ser considerado como projeto experimental e vem sendo comercializado pela IBM [/IBM 82a/, /IBM 82b/].

#### 5.8.1.2 Estruturas de dados

O SISTEMA R utiliza a abordagem relacional para estruturar os dados através de uma coleção de relações que podem ser consideradas como uma simples tabela bi-dimensional contendo um número específico de colunas e algum número de linhas não ordenadas.

O banco de dados do SISTEMA R pode ser considerado como uma coleção de relações, cada qual identificada através de um nome associado. O sistema procura proporcionar uma independência de dados isolando o usuário final, tanto quanto possível, das estruturas de armazenamento que utiliza.

#### 5.8.1.3 Linguagens hospedeiras

A linguagem SQL pode ser usada por um usuário "ad-hoc" (casual) em terminais ou por programadores, como uma sublinguagem embutida no PL/1 ou COBOL. Qualquer comando que é usado diretamente através do terminal, pode também ser embutido em um programa de aplicação.

Uma das metas básicas do SISTEMA R é, portanto, o de sustentar dois diferentes tipos de processamento sobre um banco de dados:

- a) Consultas e modificações "ad-hoc", via terminal.
- b) Consultas e modificações através de programas de aplicação escritos em COBOL ou PL/1.

Características do SQL, como comandos de consulta e modificação do banco de dados, comandos para definir e excluir objetos do banco de dados (tabelas, índices) e comandos para controlar o acesso ao banco de dados por diversos usuários são tornadas disponíveis pelo sistema em ambos os ambientes.

A linguagem alvo para a presente descrição será o PL/1.

#### 5.8.1.4 Estrutura funcional

A arquitetura funcional do SISTEMA R pode ser vista na figura 5.24.



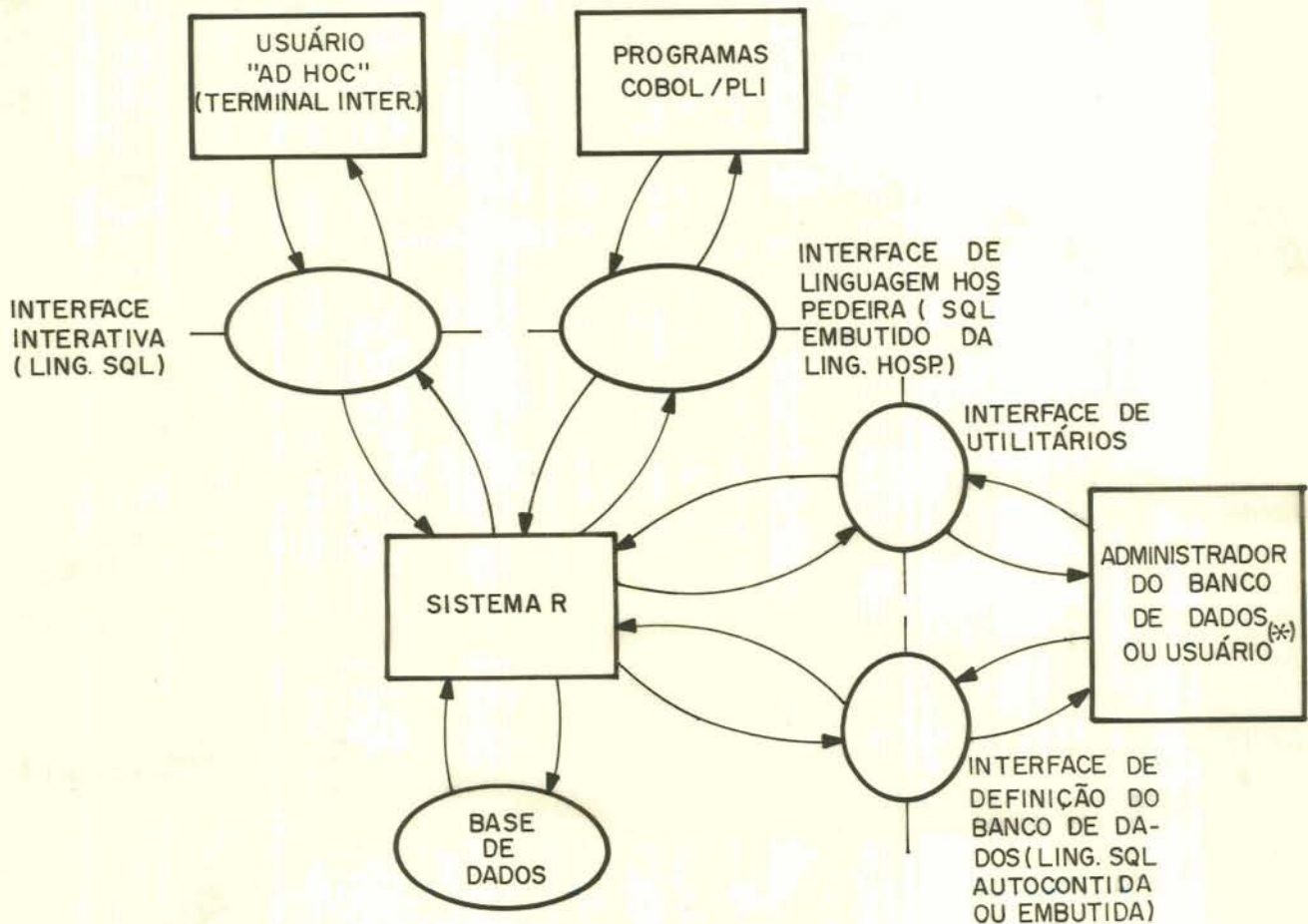


Figura 5.24 -Arquitetura funcional do SISTEMA R

(\*)- O usuário pode definir relações que ficarão sob sua responsabilidade podendo, também, atribuir o direito de uso para outras pessoas.

Nessa análise será considerada a interface de linguagem hospedeira e a de definição do banco de dados.

## 5.8.2 Interface de definição dos dados no sistema

### 5.8.2.1 Definição da base de dados

Na linguagem SQL o intercâmbio de construções entre a DDL e a DML se baseia em uma sistemática de informação mínima na DDL.

A DDL detém as funções de criação e eliminação de relações e especificar índices sobre as relações.

Ex.:

```
CREATE TABLE DEPT (DEPNRO (CHAR (5) ,NONULL) ,
                   DEPNOOME (CHAR (30) ,VAR) ,
                   DELOCAL (CHAR (30) ,VAR) )
```

Ocasionalmente torna-se necessário expandir uma relação existente adicionando novas colunas. Isto é conseguido pelo comando EXPAND.

Ex.:

```
EXPAND TABLE DEPT
      ADD COLUMN DEPQTAEMP (INTEGER)
```

O SISTEMA R não exige a figura do administrador do banco de dados, permitindo que cada usuário defina e crie componentes de seu banco de dados, com o uso do comando CREATE TABLE, tendo completo controle sobre todos os objetos por ele criados. Estes objetos podem ser criados através dos programas de aplicação.

#### 5.8.2.2 Definição de esquemas externos

Um dos aspectos mais importantes na definição de dados do SQL é a possibilidade de definir esquemas externos ("views") alternativos para armazenar dados [/CHA 75/]. No SQL esta definição é muito similar à definição de uma consulta como pode ser visto no exemplo abaixo:

```
DEFINE VIEW D50 AS
      SELECT EMPNRO, EMPNOOME, EMPFUNÇÃO
      FROM EMP
      WHERE NRODEP = 50
```

Uma importante aplicação para esquemas externos é a de permitir que um usuário acesse somente determinados dados de uma relação.

Para eliminar um esquema externo anteriormente definido usa-se o comando DROP.

Ex.: DROP VIEW D5Ø

Os esquemas externos podem, também, ser definidos dinamicamente a nível de programa de aplicação.

### 5.8.3 Aspectos funcionais do embutimento

#### 5.8.3.1 Comunicação do usuário com o banco de dados

A figura 5.25 mostra a comunicação existente entre o usuário e o banco de dados no SISTEMA R.

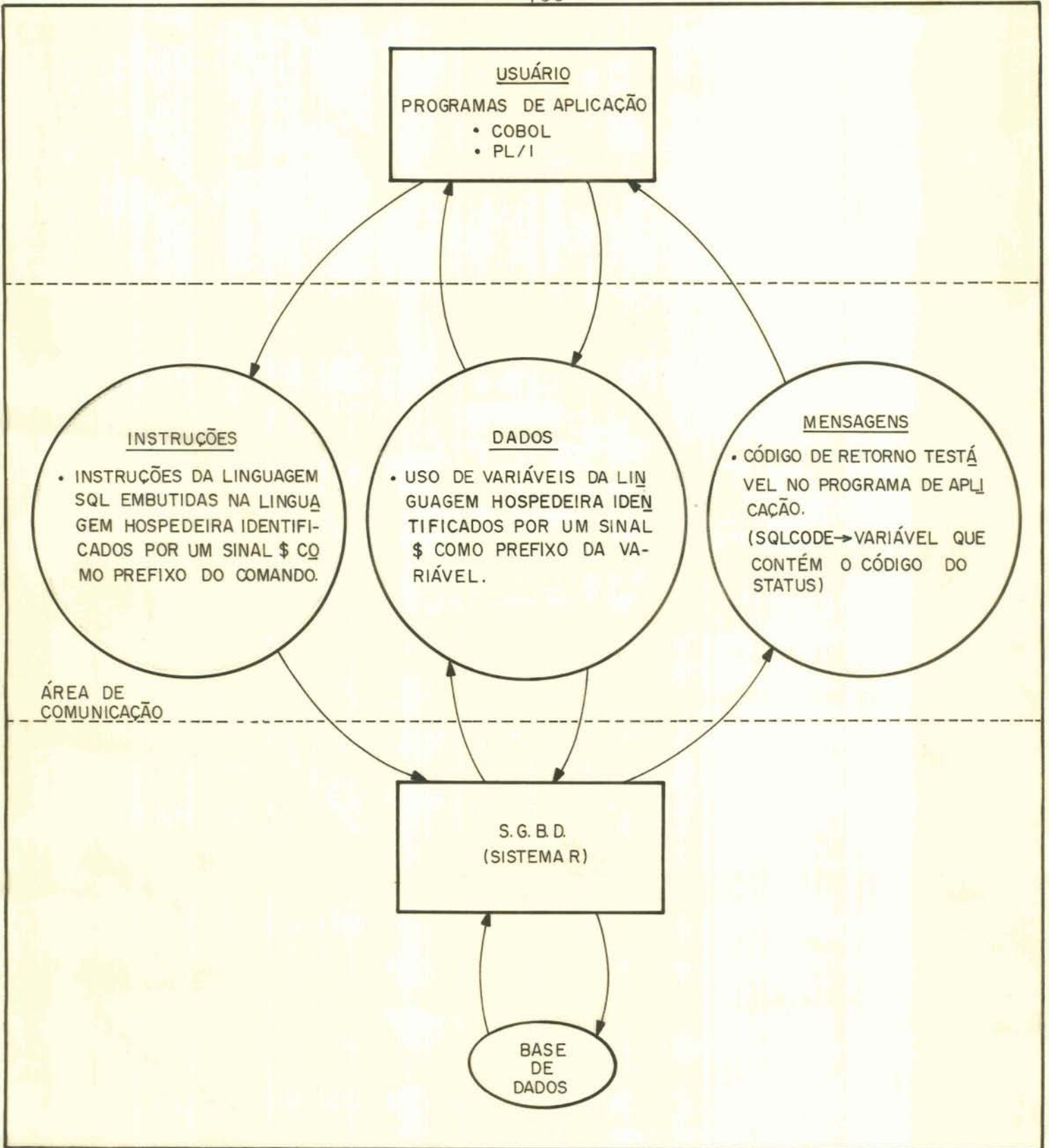


Figura 5.25- Canais de comunicação programas de aplicação e SISTEMA R

## a) DADOS

Comandos SQL em programas PL/1 ou COBOL podem conter variáveis da linguagem hospedeira se o nome da variável for precedido pelo sinal \$.

As variáveis do programa que são compartilhadas pelo programa de aplicação e o SQL são declaradas como de costume (em PL/1: na "DECLARATION SECTION" do programa; em COBOL: na "WORKING STORAGE SECTION" do programa).

Outros autores [/IBM 82a/, /IBM 82b/, /STO 77/] utilizam a simbologia de que as variáveis do programa são precedidas por ":" quando elas são referenciadas em um comando SQL embutido.

Quando uma consulta aparece em um programa PL/1 ou COBOL, ela deverá ter uma cláusula INTO contendo uma lista de variáveis do programa hospedeiro que receberão os valores dos atributos selecionados.

Ex.:

```
SELECT EMPFUNÇÃO, EMPSALARIO
      INTO $X, $Y
      FROM EMP
      WHERE EMPDEPT = 507;
```

Além destas variáveis na cláusula INTO, variáveis do programa podem aparecer em qualquer parte onde uma constante pode ser usada, em uma consulta ou qualquer outro tipo de comando SQL.

A desvantagem dos sistemas do tipo SQL é que registros, como usados no banco de dados, não são tratados como estruturas compostas no ambiente da linguagem. Por exemplo, valores de um registro do banco de dados precisam ser individualmente copiados para as variáveis do programa. Isto significa que registros não podem ser passados como argu

mento para subrotinas.

Todos os identificadores de relações e domínios a tribuídos durante a criação das relações, pelos comandos de definição, não necessitam ser declarados dentro do programa de aplicação.

#### b) INSTRUÇÕES

Um comando SQL pode ser embutido em um programa escrito na linguagem hospedeira através da colocação de um sinal \$ [/AST 76/, /CHA 80/] como prefixo do comando para distinguir este comando dos comandos da linguagem hospedeira como mostra o exemplo abaixo:

```
$UPDATE EMP SET EMPSALARIO = $X
      WHERE EMPNOME = $Y;
```

Existem literaturas [/CHA 76/, /STO 77/] onde a identificação dos comandos da linguagem SQL é realizada por um sinal \*.

Na versão SQL disponível comercialmente, o símbolo \$ foi substituído pelo prefixo "EXEC SQL" [/IBM 82a/, /IBM 82b/].

A recuperação de um conjunto de tuplas e o seu processamento no modo "uma-tupla-por-vez" é feito com o uso de "cursores".

Um cursor é um nome simbólico que o programador associa com uma consulta, através do comando LET, e que depois é referenciado para buscar uma tupla de cada vez. A especificação deste cursor pode ser vista pelo exemplo a seguir.

Ex.:

```

DCL N CHAR(30) VAR;
DCL D CHAR(5);
DCL S BIN FIXED;
$ LET C1 BE
    SELECT EMPNOME, EMPDEPT, EMPSAL
        INTO $N, $D, $S
        FROM EMP
        WHERE EMPDEPTIN
            (SELECT DEPNRO FROM DEPT
             WHERE DEPLOYAL = 'PORTO ALEGRE')
$ OPEN C1;
$ FETCH C1;

```

Todos os comandos SQL estão dispostos em formato livre.

A linguagem SQL apresenta algumas funções que podem ser usadas, em consultas, na cláusula SELECT. São elas:

- . AVG - valor médio
- . SUM - somatório
- . COUNT - contador de ocorrências
- . MAX - valor máximo
- . MIN - valor mínimo.

O SISTEMA R permite um usuário adicionar funções novas ao sistema, colocando rotinas em uma biblioteca de funções especiais.

Os componentes da tupla a serem inseridos podem ser especificados por constantes ou pelo conteúdo de variáveis do programa, não sendo obrigatória a especificação de todos os componentes. Aos componentes não especificados é atribuído um valor nulo.

Algumas facilidades da linguagem SQL em termos de potencialidades dos comandos aparecem descritas, a seguir, mediante exemplos práticos. Não é intenção, no entanto, des

crever os detalhes de todos os comandos disponíveis.

Uma relação pode ser particionada em grupos, de a cordo com valores de alguns atributos, e então, uma função aplicada a cada grupo.

Ex.: Listar todos os departamentos e o salário mé dio de cada.

```
SELECT EMPDEPT, AVG (EMPSALARIO)
FROM EMP
GROUP BY EMPDEPT
```

A uma relação particionada em grupos pode ser a- plicado um predicado ou conjunto de predicados para esco- lher somente certos grupos e desqualificar outros. Esta qua- lificação é conseguida através da cláusula HAVING.

Ex.: Listar os departamentos nos quais o salário médio dos empregados é menor que 10000.

```
SELECT EMPDEPT
FROM EMP
GROUP BY EMPDEPT
HAVING AVG (EMPSALARIO) < 10000
```

Os operadores de interseção (intersect), união (union) e diferença (minus) também estão disponíveis no SQL.

Ex.: Listar todos os departamentos que não têm em pregados.

```
SELECT DEPNRO
FROM DEPT
MINUS
SELECT EMPDEPT
FROM EMP
```



O SQL detém, também, a facilidade de combinar comandos para, por exemplo, fazer a inclusão de múltiplas tuplas em uma relação através de uma única invocação.

```
Ex.: INSERT INTO TEMP
      SELECT *
      FROM EMP
      WHERE EMPIDADE > 25
```

OBS.: O asterisco indica que todos os campos serão considerados.

Manipulação de múltiplas tuplas pode também ser realizada pelos comandos UPDATE (alteração) e DELETE (exclusão).

O SQL proporciona uma facilidade especial, através do comando EXECUTE, que possibilita um programa hospedeiro sustentar usuários interativos. O programa, em tempo de execução, lê de um terminal, um comando SQL, a ser executado, para dentro de uma variável da linguagem hospedeira.

```
Ex.: $ EXECUTE QSTRING;
```

Em resposta ao \$ EXECUTE, o comando SQL armazenado em QSTRING é analisado, um caminho de acesso é selecionado para ele, e então é traduzido para uma rotina em linguagem de máquina e executado.

Ocasionalmente é necessário para um programa executar um comando SQL que não é conhecido de antemão. O exemplo clássico é o programa que realiza a carga de um arquivo (atualização).

Para resolver este problema a solução é desmembrar a função \$ EXECUTE QSTRING em 2 comandos chamados \$PREPARE e \$EXECUTE.

```
Ex.: $PREPARE S1 AS QSTRING
```

A função deste comando é analisar o comando SQL encontrado em QSTRING, localizar o caminho de acesso para ele, traduzi-lo para uma rotina em linguagem de máquina e associar esta rotina com o nome S1. No lugar de valores de dados específicos os parâmetros podem ser denotados por "?". Posteriormente, o seguinte comando pode ser executado:

```
$ EXECUTE S1 USING $X, $Y, $Z
```

Para sintetizar as características das instruções da linguagem SQL, poderiam ser enumeradas as seguintes facilidades:

- . Mapeamento simples: mapeamento que retorna os valores dos dados em alguma coluna que é associada com um valor de dado conhecido em outra coluna.

- . Seleção (todos dados num registro): para selecionar todos os valores de dados em um registro.

- . Projeção (todos dados em uma coluna): para selecionar todos os valores de dados em uma coluna.

- . Atribuição: permite ao usuário atribuir resultados da consulta a novas tabelas (INSERT combinado com SELECT).

- . Funções: permite a aplicação de funções matemáticas tais como SUM, COUNT, MAX, MIN, AVG.

- . Conjunção e disjunção: AND/OR.

- . Operações sobre conjuntos: operações como união, intersecção, diferença.

- . Combinação (aninhamento): a linguagem permite que o resultado de um mapeamento seja usado como entrada de outro.

- . Agrupamento (group by): Organização de registros de uma tabela em grupos por casamento de valores.

Complementando a análise dos aspectos vinculados às instruções DML, no ambiente da linguagem hospedeira, as funções dos principais comandos do SQL são mostradas na figura 5.26, segundo a versão da linguagem atualmente sendo comercializada pela IBM [/IBM 82a/].

TIPO DE COMANDO	COMANDO	FUNÇÃO
CONSULTA	SELECT	- RECUPERA DADOS DE UMA OU MAIS TABELAS
MANIPULAÇÃO	INSERT UPDATE DELETE	- INCLUSÃO DE UMA OU MAIS TUPLAS - MODIFICAÇÃO DE DADOS EM UMA OU MAIS TUPLAS - EXCLUSÃO DE UMA OU MAIS TUPLAS
DEFINIÇÃO	CREATE TABLE DROP TABLE ALTER TABLE (EXPAND) CREATE INDEX DROP INDEX CREATE SYNONYM DROP SYNONYM CREATE VIEW DROP VIEW	- DEFINE A ESTRUTURA DE NOVA TABELA - DESTRÓI UMA TABELA - EXPANSÃO DA TABELA EM TERMOS DE COLUNAS - DEFINE UM ÍNDICE PARA ACESSO EM DETERMINADA SEQUÊNCIA. - DESTRÓI UM ÍNDICE - DEFINE UM NOME ALTERNATIVO PARA A TABELA - DESTRÓI A DEFINIÇÃO DE UM SINÔNIMO - DEFINE UMA "VIEW" - DESTRÓI UMA "VIEW"
AUTORIZAÇÃO	GRANT REVOKE	- FORNECE PRIVILÉGIOS A UM OU MAIS USUÁRIOS - RETIRA O PRIVILÉGIO DE UM OU MAIS USUÁRIOS
CONTROLE DE DECLARAÇÃO	BEGIN DECL. SECTION END DECLARE SECTION	- INÍCIO DO CONJUNTO DE DECLARAÇÃO DAS VARIÁVEIS DO PROGRAMA HOSPEDEIRO USADA NOS COMANDOS SQL - FIM DAS DECLARAÇÕES DAS VARIÁVEIS HOSPEDEIRAS
CONTROLE DE CURSOR	DECLARE (LET) OPEN FETCH CLOSE	- DEFINE UM CURSOR - INICIA A OPERAÇÃO DE RECUPERAÇÃO COM BASE NO CURSOR - RECUPERA A PRÓXIMA TUPLA - FINALIZA O USO DO CONJUNTO DE TUPLAS RECUPERADAS COM BASE NO CURSOR
MANIPULAÇÃO DE COMANDOS DINAMICAMENTE ESPECIFICADOS	PREPARE DESCRIBE EXECUTE	- PASSA UM CONJUNTO DE CARACTERES PARA O SQL A FIM DE QUE SEJA PRÉ-PROCESSADO COMO COMANDO - RECUPERA INFORMAÇÕES SOBRE UM COMANDO SQL SUBMETIDO AO "PREPARE" - EXECUTA O COMANDO SUBMETIDO AO "PREPARE"
ESTRUTURA DO CÓDIGO DE RETORNO	INCLUDE WHENEVER	- INTRODUZ A ESTRUTURA DO CÓDIGO DE RETORNO (ÁREA - SQLCA ; CAMPO SQLCODE) NO PROGRAMA - PERMITE A MANIPULAÇÃO DE EXCEÇÕES SEM O TESTE EXPLÍCITO DO CÓDIGO DE RETORNO

Figura 5.26 : Quadro de instruções DML do SQL

## c) MENSAGENS

Para cada comando SQL embutido executado [/IBM 82a/], o sistema retorna código de informação ao programa relativo ao sucesso ou falha na execução do comando.

O programa não necessita usar esta informação, mas deve reservar uma área para isto, chamada "SQL COMMUNICATIONS AREA - SQLCA".

Esta estrutura é incluída no programa através do comando INCLUDE SQLCA.

Na estrutura do código de retorno existe um campo importante denominado SQLCODE. Este campo contém um inteiro que sumariza o estado do comando SQL executado. Em geral SQLCODE=0 denota sucesso.

Opcionalmente o usuário pode especificar a cláusula WHENEVER que permite a manipulação de exceções sem o teste explícito do código de retorno.

## 5.8.3.2 Conversões

Todas as conversões de tipo são realizadas implicitamente pelo sistema, ou seja, o sistema se encarrega de efetuar as conversões necessárias para que os dados armazenados no banco de dados sejam passados de forma coerente para as variáveis do programa de aplicação sem nenhuma intervenção do usuário [/STO 77/].

## 5.8.3.3 Estruturas de controle de fluxo

Conforme visto anteriormente, as consultas realizadas, quando o SQL é usado embutido em uma linguagem hospedeira, necessitam da especificação de um "cursor" que estará associado a esta consulta. Neste caso o comando DML de

consulta assinalará todas as tuplas que satisfizerem o critério de seleção. No entanto, para trazer as tuplas, individualmente, para a área do programa de aplicação, o usuário deverá providenciar a codificação de procedimentos, usando comandos da linguagem hospedeira, que proporcionarão a estruturação de um fluxo de controle de iteração. Cada tupla será individualmente recuperada pelo comando "FETCH" e um teste explícito do conteúdo do código de estado retornado indicará o momento de finalizar o laço ("loop") de recuperação.

Ex.:

```
$ OPEN C1;
  DO WHILE (SQLCODE = 0);
$   FETCH C1;
     DISPLAY (N,D,S)
  END;
```

#### 5.8.3.4 Considerações sobre proteção de dados

A segurança do sistema baseia-se num esquema de autorizações que delineiam o perfil do usuário apto a utilizar as facilidades que o sistema oferece.

O SISTEMA R permite uma instalação de banco de dados controlar que usuários que estão autorizados a criar novas relações no banco de dados. A pessoa responsável pelo controle de autorizações terá privilégios especiais. Isto permite que uma instalação escolha um controle centralizado, em que todas estruturas do banco de dados e privilégios sejam controlados por um único administrador de banco de dados, ou um controle descentralizado no qual cada usuário pode controlar suas próprias relações.

Para autorizar um determinado usuário a usar uma relação o SQL tem disponível o comando GRANT.

Ex.:

```
GRANT READ, INSERT, UPDATE (FUNÇÃO, NRODEP)
  ON EMP TO FULANO, CICLANO
  WITH GRANT OPTION
```

E para revogar a autorização usa-se o comando REVOKE.

Ex.:

```
REVOKE UPDATE ON EMP FROM FULANO
```

Outra característica da linguagem relacionada com aspectos de segurança é, como já mencionado anteriormente, o uso de esquemas externos para limitar o acesso a determinados dados de uma relação.

O controle de integridade é conseguido através de transações, asserções sobre integridade e gatilhos (triggers).

Convém salientar, no entanto, que as asserções sobre integridade e gatilhos são facilidades contidas nas especificações iniciais do projeto da linguagem SQL[/CHA 76/] mas que na atual versão, sendo comercializada pela IBM, ainda não foram implementadas [/IBM 82a/].

Uma transação é uma seqüência de chamadas a interface do banco de dados (RDI) que devem ser executadas como uma ação atômica (indivisível). É especificada com o uso dos operadores BEGIN TRANSACTION e END TRANSACTION. O operador SAVE pode ser usado dentro de uma transação, demarcando pontos de recuperação. Diversos pontos de recuperação podem ser declarados em uma transação. O operador RESTORE permite a volta ao início da transação (se não for especificado ponto de recuperação) ou a qualquer um de seus pontos de recuperação, sendo restauradas todas as alterações feitas no banco de dados, bem como o estado de cada um dos cursores usados na transação. O comando RESTORE não tem efeito sobre

alterações feitas por outros usuários.

O mecanismo de gatilho permite a especificação de comandos que devem ser executados sempre que ocorra um determinado evento. O evento pode ser uma consulta, inserção, exclusão ou alteração em uma particular tabela ou esquema externo. Se um comando SQL executa uma ação sobre diversas tuplas e esta ação invoca um gatilho, este será executado repetidamente após a ação sobre cada tupla.

Ex.:

```

DEFINE TRIGGER T1
  ON UPDATE OF EMP (EMPDEPT) :
    (UPDATE DEPT
     SET DEPQTAEMP = DEPQTAEMP+1
     WHERE DEPNRO = NEW EMP.EMPDEPT;
     UPDATE DEPT
     SET DEPQTAEMP = DEPQTAEMP-1
     WHERE DEPNRO = OLD EMP.EMPDEPT)

```

Um usuário que controla as autorizações pode eliminar um gatilho por meio do comando DROP TRIGGER.

Asserções de integridade podem ser especificadas sobre tuplas individuais ou sobre conjuntos de tuplas. Uma asserção é aceita somente se, no momento de sua especificação, é verdadeira para o banco de dados. Os comandos ASSERT e DROP ASSERTION permitem a especificação e a eliminação de asserções. Asserções de integridade são normalmente suspensas em uma transação. No fim da transação todas asserções relevantes são verificadas, e, se alguma violação ocorreu, toda a transação é rejeitada. Com isto pode haver uma inconsistência temporária do banco de dados. Para contornar este problema o usuário pode, alternativamente, definir uma asserção como IMMEDIATE. Neste caso é verificada após cada alteração dos dados, caso contrário, é verificada somente após a efetivação de uma transação completa.



Ex:

```
ASSERT A1 IMMEDIATE
ON UPDATE OF EMP(EMPSALARIO):
NEW EMPSALARIO > = OLD EMPSALARIO
```

A proteção no SISTEMA R é conseguida através do mecanismo de bloqueio (LOCKING). A gerência deste mecanismo, tanto lógico quanto físico, é feita pelo sistema de armazenamento (RSS).

Usualmente um comando SQL é a unidade para o qual bloqueios atuam. Entretanto, chaveamentos podem também ser declarados para seqüência de comandos através das especificações "BEGIN-TRANSACTION" e "END-TRANSACTION".

#### 5.8.3.5 Considerações sobre homogeneidade

A homogeneidade no SQL pode ser constatada pelo fato de comandos fontes utilizados para realizar uma consulta por um usuário num terminal interativo são sintaticamente idênticos aos comandos utilizados no programa de aplicação escrito na linguagem COBOL ou escrito na linguagem PL/1.

Com isto os programadores são capazes de depurar facilmente seus programas de aplicação, submetendo as consultas interativas contra o banco de dados para observar os resultados de seus programas escritos em uma das linguagens hospedeiras, sem necessitar para isto estudar uma sintaxe diferente.

Existe, portanto, uniformidade sintática entre os comandos fontes do SQL embutidos em qualquer uma de suas linguagens hospedeiras. No entanto, analisando a existência de homogeneidade entre a DML e os comandos da linguagem hospedeira, percebe-se que a DML segue padrões próprios que são independentes dos padrões da linguagem hospedeira.

#### 5.8.4 Aspectos de implementação do embutimento

##### 5.8.4.1 Descrição da forma de embutimento

Um programa PL/1 ou COBOL que tem embutido comandos SQL é submetido ao precompilador do SISTEMA R. Este precompilador tem a função de "filtro" no qual todos os comandos SQL são reconhecidos e substituídos por chamadas a um módulo que executará o comando.

O sistema de dados relacional (RDS) tem 2 funções distintas [/AST 76/]:

. Um precompilador, chamado XPREP que é usado para precompilar programas da linguagem hospedeira e instalando-os como "PROGRAMAS ENLATADOS" sob o SISTEMA R (figura 5.27).

. Um sistema de execução, chamado XRDI, que controla a execução do "PROGRAMA ENLATADO" e também executa comandos SQL para usuários "ad-hoc" (figura 5.28).

Quando um programa de aplicação tem escrito um programa PL/1 ou COBOL com comandos SQL embutidos, seu primeiro passo é submeter o programa ao precompilador XPREF. Este localiza os comandos SQL no programa e os traduz em um módulo de acesso em linguagem de máquina. No programa do usuário os comandos SQL são substituídos por chamadas ao módulo de acesso. O módulo de acesso é armazenado no banco de dados do SISTEMA R para protegê-lo de modificações não autorizadas.

As vantagens advindas da precompilação são:

. Muitas das tarefas de análise sintática, interligação de nomes, seleção de caminho de acesso e verificação de autorizações podem ser feitas pelo precompilador tirando esta tarefa do processo de execução.

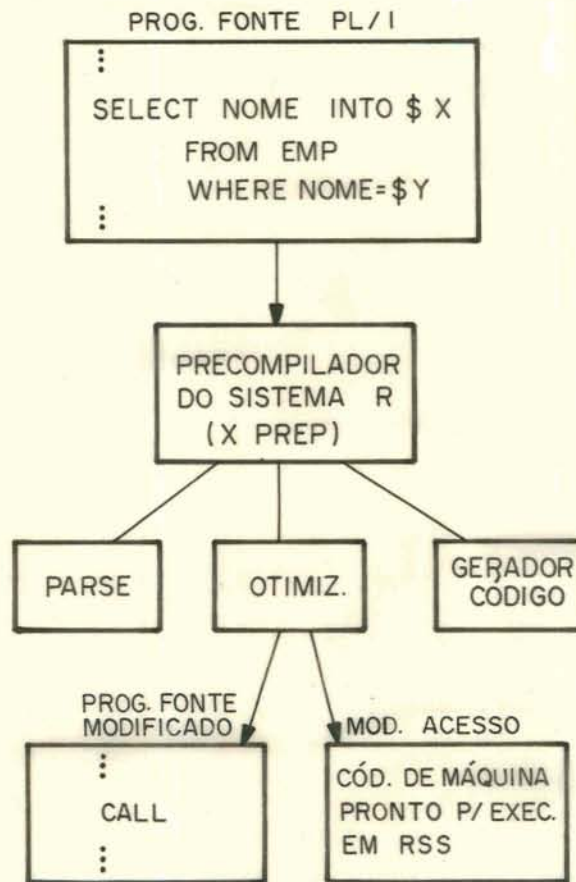


Figura 5.27- Passos da precompilação no SQL

. O módulo de acesso, por ser montado para um programa específico, é muito menor e é mais eficiente do que um interpretador SQL geral.

Depois da precompilação, o programa do usuário contém somente comandos PL/1 ou COBOL e pode ser compilado usando um compilador de linguagem convencional.

Quando um "programa enlatado" é executado no SISTEMA R, ele realiza chamadas ao XRDI, que por sua vez carrega e invoca o módulo de acesso que opera sobre o banco de dados pela realização de chamadas ao RSS e libera o resulta

do ao programa do usuário.

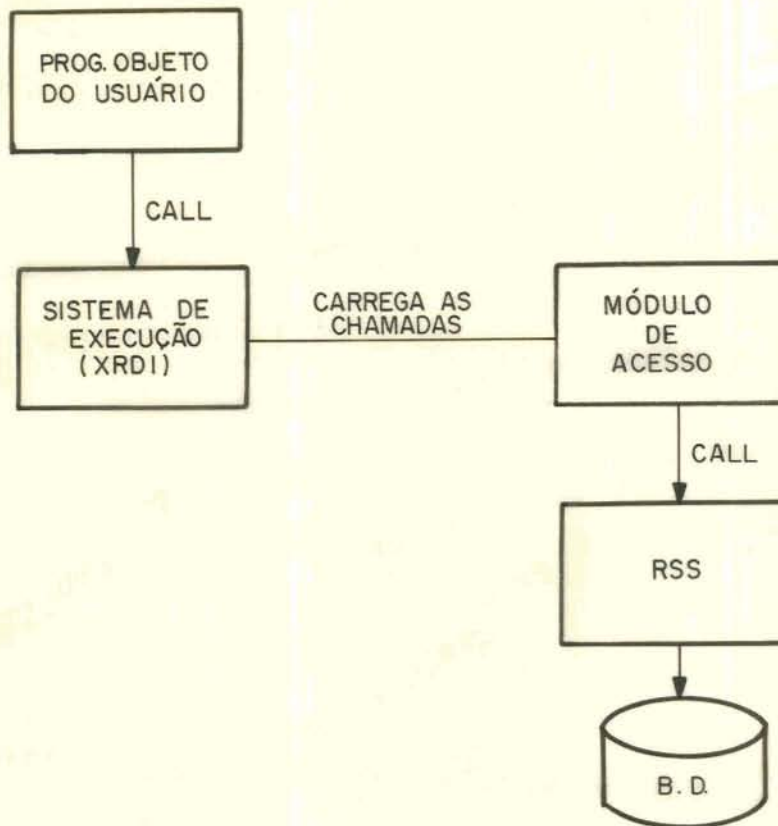


Figura 5.28 - Passos da execução no SQL

Quando um programa PL/1 ou COBOL com comandos SQL embutidos é apresentado ao precompilador do SISTEMA R, ele examina cuidadosamente o programa para achar os comandos SQL e substitui cada comando SQL por uma chamada válida da linguagem hospedeira. Além disso, cada comando SQL é submetido a um processo de 3 passos a fim de traduzi-lo em uma rotina em linguagem de máquina.

Os 3 passos são os seguintes:

a) Análise sintática

O analisador sintático verifica o comando SQL para validade sintática e o traduz em uma representação em árvore de análise sintática (PARSE-TREE) convencional. O analisador também retorna ao precompilador do SISTEMA R, 2 listas de variáveis do programa hospedeiro encontradas no comando SQL: uma lista de variáveis de entrada (valores fornecidos pelo programa chamador e usados na execução do comando) e uma lista de variáveis de saída (posições visadas para os dados a serem carregados pelo comando). Por exemplo, se o comando SQL entregue ao analisador sintático for o seguinte:

```
SELECT NOME, SALARIO
INTO $X, $Y
FROM EMP
WHERE GERENTE = $A AND FUNCAO = $B
```

as variáveis de entrada seriam A e B e as de saída seriam X e Y.

b) Otimizador

O otimizador usa a "PARSE-TREE" como entrada e executa as seguintes tarefas:

. Usando catálogos internos do SISTEMA R, ele resolve todos nomes simbólicos encontrados no comando SQL para os objetos internos do banco de dados (construções).

. Uma verificação é feita para garantir que o usuário corrente está autorizado para executar a operação indicada sobre a tabela.

. Se o comando SQL opera sobre um ou mais esquemas externos definidos pelo usuário, as definições são mergeadas com o comando SQL para formar uma nova "parse tree" SQL composta que opera sobre tabelas reais armazenadas

das.

. O otimizador usa os catálogos do sistema para achar o conjunto de índices disponíveis e outras informações estatísticas sobre as tabelas a serem processadas. Esta informação é usada para escolher o caminho de acesso e um algoritmo para processamento do comando SQL. O otimizador representa sua escolha do caminho de acesso por meio de uma ASL (Access Specification Language).

#### c) Gerador de código

O gerador de código traduz as estruturas ASL, produzidas pelo otimizador, em uma rotina em linguagem de máquina que implementa a escolha do caminho de acesso. Esta rotina em linguagem de máquina é denominada "seção". Quando executando, a seção acessará o banco de dados através do uso da lista de parâmetros da RSS que foram produzidos pelo otimizador.

Depois de todos os comandos SQL em um programa tiverem sido traduzidos em seções, as seções são coletadas juntas para formar um módulo de acesso. Aliado ao código de linguagem de máquina, cada seção mantém o comando SQL do qual ela foi originalmente construída, assim possibilitando a seção ser reconstruída se seu caminho original de acesso vir a ser ineficaz.

#### 5.8.4.2 Considerações sobre a portabilidade da linguagem

Como visto anteriormente, o SISTEMA R implementa o embutimento da linguagem SQL nas linguagens hospedeiras PL/1 ou COBOL através do uso de um precompilador que desempenha um papel de filtro, identificando e traduzindo os comandos SQL embutidos no programa fonte da linguagem hospedeira.

Para que o embutimento seja implementado em outras linguagens hospedeiras seria necessário levar em consideração os seguintes requisitos e necessidades:

. A nova linguagem hospedeira deve possuir, em sua estrutura, facilidades para chamadas externas a subrotinas (comando CALL).

. O precompilador deve ser adaptado para que converta os comandos SQL em chamadas a subrotinas (módulos de acesso ao banco de dados) nos padrões da nova linguagem hospedeira.

## 5.9 PASCAL/R

### 5.9.1 Introdução

Tendo em vista que não se pode considerar que o PASCAL/R se enquadre perfeitamente, com suas características, numa abordagem de um sistema de gerência de banco de dados, será realizada uma análise voltada às principais facilidades da linguagem em termos de estruturas e construções que implementam o tratamento de relações e primitivas para banco de dados a fim de que se possa, posteriormente, utilizá-las em comparação com as características de uma linguagem de manipulação de dados embutida em programas de aplicação através da interface com linguagem hospedeira.

O PASCAL/R [/SCH 77/, /SCH 80/] é uma linguagem convencional que foi estendida para permitir o uso de relações e primitivas de banco de dados.

Não há, portanto, linguagens hospedeiras vinculadas e nem a necessidade de estabelecer canais de comunicação que normalmente aparecem nas linguagens de manipulação de dados para manter um fluxo de compartilhamento de informações entre o programa de aplicação e o sistema de gerência de banco de dados.

Tradicionalmente um banco de dados, com sua linguagem de manipulação de dados associada, é visto como um sistema independente sendo que os dados estão interligados com o usuário através de interfaces fixas na forma de áreas de entrada/saída. Problemas que surgem da integração das construções da linguagem de banco de dados com as das linguagens convencionais de alto nível, como por exemplo o relacionamento entre as estruturas de dados, têm sido alvo de inúmeras investigações com vistas a aperfeiçoar as linguagens existentes e para favorecer o desenvolvimento de concepções de banco de dados. O resultado de algumas das investigações feitas, neste sentido, deu origem à implemen-



tação do PASCAL/R.

O sistema PASCAL/R é considerado uma estrutura de trabalho dentro da qual as concepções essenciais de linguagens de programação e bancos de dados se unificam resultando num maior grau de compreensão e aprendizado por parte das pessoas que dela fazem uso para as suas aplicações particulares.

### 5.9.2 Estrutura de dados

O PASCAL/R estende a linguagem PASCAL principalmente no que se refere à implementação de relações. Um dos principais objetivos do projeto do PASCAL/R foi o de integrar as estruturas relacionais aos dados do PASCAL e às estruturas de controle de fluxo de programas.

Com isto, as estruturas de dados na nova linguagem passaram a representar:

- . Construções originais da linguagem PASCAL.
- . Novas construções implementadas no projeto "PASCAL/R".

#### 5.9.2.1 Retrospectiva sobre as construções PASCAL

Na linguagem PASCAL, um algoritmo ou um programa consiste de uma descrição das ações a serem executadas (statements) e uma descrição dos dados que são manipulados por estas ações (declarations ou definitions).

Os dados são representados por valores de variáveis que devem ser introduzidas no programa por uma declaração de variável a qual faz a associação de um identificador a um determinado tipo de dado. A sintaxe é a seguinte:

```
VAR <identificador(es)>:<tipo de dado>;
```

O tipo do dado essencialmente define o conjunto de valores que podem ser assumidos pela variável. Um tipo de dado pode no PASCAL ser descrito diretamente na declaração da variável ou então pode ser referenciado por um identificador de tipo (definição explícita de tipo), como poderá ser visto pela sintaxe apresentada abaixo:

```
TYPE <identificador> = <tipo>;
```

Os tipos de dados básicos são: SCALAR (conjunto ordenado de valores), BOOLEAN, INTEGER, CHAR, REAL e SUB-RANGE (subconjunto de um tipo scalar indicando o menor e maior valor).

Além dos tipos básicos, o PASCAL ainda admite tipos estruturados que são definidos pela descrição dos tipos de cada um de seus componentes e pela indicação de um método de estruturação. Os vários métodos diferem no mecanismo de seleção que serve para selecionar os componentes de uma variável do tipo estruturado. Tem-se assim os seguintes métodos de estruturação disponíveis no PASCAL:

- . estrutura tipo array
- . estrutura tipo registro
- . estrutura tipo conjunto (SET)
- . estrutura tipo arquivo (FILE).

Com respeito a comandos a linguagem PASCAL admite comandos de atribuição, comandos de seleção do tipo "IF-THEN-ELSE" e "CASE OF", comandos de iteração do tipo "WHILE DO", "REPEAT UNTIL" e "FOR TO DO" além de comandos compostos ("BEGIN-END") e também os de entrada/saída ("READ" & "WRITE"). Admite também rotinas do tipo "PROCEDURE" e funções.

Nas expressões pode haver operadores aritméticos, booleanos, de conjuntos (união, interseção e diferença) e relacionais (igualdade, desigualdade, está contido, ...).

### 5.9.2.2 Novas construções implementadas

A extensão da linguagem PASCAL resultando no PASCAL/R proporcionou a implementação de novas construções na linguagem, admitindo novos métodos de estruturação dos dados.

O PASCAL/R proporciona 2 métodos adicionais:

- . estrutura tipo relação
- . estrutura tipo banco de dados.

Uma relação é composta por um conjunto de tuplas do tipo registro (record) enquanto que um conglomerado de relações constitui a estrutura "banco de dados".

### 5.9.3 Definição dos dados

Na presente descrição foi considerada somente a definição dos dados relativos às novas estruturas implementadas no projeto do PASCAL/R.

#### 5.9.3.1 Estruturas tipo relação

A definição do tipo de dado relação é baseada na definição do tipo de dado registro (record). O valor de uma variável do tipo relação é um conjunto de tuplas, cada qual do tipo de registro apresentado na definição do tipo relação.

Em uma estrutura do tipo relação, todos os elementos são do mesmo tipo e são identificados univocamente pela chave primária definida. Uma lista de chaves caracteriza uma parte da tupla através da enumeração dos componentes identificadores dos campos.

O formato geral de uma especificação de estrutura do tipo relação é:

```
<definição tipo relação>:: =
  TYPE< nome tipo relação >= RELATION "<" <chave prim.> ">"
    OF <nome tipo registro>
```

Exemplo:

```
TYPE REGEMP = RECORD
    EMPNRO      : INTEGER;
    EMPNOME     : ALPHA;
    EMPDATAADM : DATA;
    :
    EMPFUNCAO  : CARGO
END
RELEMP = RELATION <EMPNRO> OF REGEMP ;
VAR EMPREGADO = RELEMP;
```

### 5.9.3.2 Estruturas tipo banco de dados

Uma estrutura do tipo banco de dados consiste de um número fixo de componentes do tipo relação (possivelmente de diferentes tipos). Para cada relação componente do banco de dados é especificado o seu tipo, através de uma definição de tipo ou através de um identificador de tipo previamente definido.

Cada item pertencente a uma relação pode ser individualmente referenciado através dos identificadores montados em forma hierárquica onde os identificadores dentro da hierarquia são separados por um ponto (".").

O formato geral de uma especificação de estrutura do tipo banco de dados é:

```
<definição tipo banco de dados>:: =
  TYPE <nome tipo BD> = DATABASE <lista de relações>
  END
```

Exemplo:

```
TYPE PESSOAL = DATABASE
    EMPREGADO : RELEMP;
    DEPARTAMENTOS: RELDEP;
    VENCEMP : RELVENC;
    TABELAVENC : RELATION <TABNRO> OF
        RECORD
            TABNRO: INTEGER;
            TABNOME: ALPHA;
            TABVAL: INTEGER
    END
END
VAR EMPPESSOAL = PESSOAL;
```

Para referência a uma relação integrante, a especificação seria, por exemplo:

```
EMPPESSOAL.EMPREGADO
```

#### 5.9.4 Manipulação dos dados

O projeto do PASCAL/R procurou considerar construções organizadas em torno de relações e, para tal, a implementação envolveu:

- . Instruções primitivas para alterar relações a nível de tuplas individuais.

- . Facilidades de recuperação operando sobre relações a nível de conjunto de tuplas.

Com estes recursos muitas tarefas de programação



A variável de controle do registro é do mesmo tipo do registro que compõe a relação. A variável "RANGE" da relação é uma variável de tipo relação e serve para determinar qual a relação alvo da pesquisa. Esta variável pode estar associada a determinado banco de dados (relação contida no B.D.) e neste caso deverá ser referenciada, indicando-se qual o banco de dados.

A expressão lógica que pode participar na seleção tem os usuais operadores lógicos e relacionais e pode conter também quantificadores. Como operandos, a expressão lógica pode conter componentes da variável de controle do registro, variáveis dos predicados, variáveis do programa ou constantes.

Um exemplo prático, com base nos tipos definidos anteriormente (REGEMP, RELEMP) ilustra o uso de expressões:

```

:
VAR VREGEMP          : REGEMP;
    EMPREGADO,RELRESUL: RELEMP;

:
RELRESUL:= [EACH VREGEMP IN EMPREGADO:VREGEMP.EMPFUNÇÃO = PRO
                                                    GRAMADOR]

```

Esta expressão seria usada para selecionar cada registro "VREGEMP" na relação "EMPREGADO" que tivesse no atributo "EMPFUNÇÃO" o conteúdo programador. O resultado desta expressão é um conjunto de registros que formarão uma nova relação.

Como pode ser constatado, portanto, em uma única expressão, é possível selecionar e recuperar um conjunto de registros (tuplas) pertencentes a uma relação, possibilitando a construção de subrelações.

O PASCAL/R admite, também, expressões com quanti-

ficadores (predicados sobre relações) e o formato segue o seguinte padrão:

```

<predicado> ::= <quantificador><var.controle reg>
                IN <var. "range" rel>
                ( <expressão lógica> )
<expressão lógica> ::= <termo> | <termo><oper-lógico>
                        <expressão lógica>
<quantificador > ::= SOME/ALL
<oper.lógico > ::= AND/OR
<termo > ::= <predicado> | <variável><oper-relação>
                <var/const>
<oper-relação > ::= "=" | "<" | ">" | "≠" | "≤" | "≥"

```

Para ilustrar o uso de expressões com quantificadores será apresentado um exemplo comparativo com uma expressão de relação.

A expressão de relação

```
Rel1 <= Rel2
```

É equivalente à expressão com quantificadores

```
ALL Reg1 IN Rel1 SOME Reg2 IN Rel2 (Reg1<=Reg2)
```

#### 5.9.4.2 Instruções de manipulação de relações como um todo

Estas instruções aparecem sob a forma de comandos de atribuição.

Os comandos de atribuição servem para substituir o valor corrente de uma variável por um novo valor especificado por meio de uma expressão.

Para o PASCAL/R foram adicionados comandos de atribuição que atualizam uma variável tipo relação por intermédio de uma expressão de relação, usando os operadores



de atualização de relação (:+, :-, :&).

O valor alterado de uma relação pode ocorrer através de inserção, exclusão ou modificação de tuplas. Estas operações não são de fato elementares pois através delas um conjunto de tuplas podem ser alteradas.

. Operador de inserção: :+

Ex.: Rel1 :+ Rel2

Esta instrução poderia também ser expressa da seguinte forma:

```
Rel1:= [EACH Reg1 IN Rel1:TRUE, EACH Reg2 IN Rel2:
        NOT SOME Regaux IN Rel1 (Reg2.CHAVE = Regaux.CHAVE)]
```

Em Rel1 é adicionado uma cópia de todas as tuplas de Rel2 cujo valor de chave originalmente não constava em Rel1, sendo que Rel1 e Rel2 devem ser do mesmo tipo.

. Operador de exclusão: :-

Ex.: Rel1 :- Rel2

Usando quantificadores a instrução poderia ser escrita como segue:

```
Rel1:= [EACH Reg1 IN Rel1:
        NOT SOME Reg2 IN Rel2 (Reg.1.CHAVE = REG2.CHAVE)]
```

Em Rel1 são excluídas todas as tuplas cujo valor de chave consta nas tuplas da relação Rel2.

. Operador de modificação: :&

Ex.: Rel1 :& Rel2

Esta instrução é equivalente à:

```

Rel1:= [EACH Reg1 IN Rel1:
        NOT SOME Reg2 IN Rel2 (Reg1.CHAVE = Reg2.CHAVE),
        EACH Regaux2 IN Rel2:
        SOME Regaux1 IN Rel1 (Regaux1.CHAVE =
                               Regaux2.CHAVE) ]

```

Em Rel1 são modificadas todas as tuplas cujo valor de chave consta nas tuplas da relação Rel2.

. Operador de atribuição: :=

```

Ex.: Rel1 := Rel2
ou   Rel1 := [EACH Reg2 IN Rel2]

```

Rel1 recebe todas as tuplas contidas em Rel2.

Pode-se, também, adicionar registros a uma relação através dos comandos de atribuição e além disso é possível inicializar uma relação a partir de um valor vazio.

A relação vazia é denotada por [ ].

A adição de registros a uma relação pode ser vista no exemplo abaixo:

```

:
VAR VREGEMP : REGEMP;
    EMPREGADO:RELEMP;
VREGEMP.EMPNO := '1200';
VREGEMP.EMPNO := 'JOAO DA SILVA';
VREGEMP.EMPFUNCAO := 'PROGRAMADOR';
EMPREGADO := [ ];
EMPREGADO :+[VREGEMP];
:

```

#### 5.9.4.3 Instruções de manipulação de relações no modo "uma-tupla-por-vez"

Estas instruções aparecem sob a forma de primitivas de tratamento de relações. Estas rotinas selecionam uma tupla específica de uma relação indicada devolvendo-a em uma relação resultante, ambas passadas como parâmetro na chamada. A relação Rel é indicada como primeiro parâmetro e se o elemento procurado existir, este é colocado no segundo parâmetro, Reltupla, que deve ser de mesmo tipo da primeira.

As rotinas são as seguintes:

- LOW(Rel,Reltupla) - Seleciona o elemento da relação que tem o menor valor de chave.
- NEXT(Rel,Reltupla) - Seleciona o elemento da relação que tem o valor de chave seguinte ao valor corrente na variável Reltupla.
- THIS(Rel,Reltupla) - Seleciona o elemento com chave igual ao informado na variável Reltupla.
- HIGH(Rel,Reltupla) - Seleciona o elemento da relação que tem o maior valor de chave.
- PRIOR(Rel,Reltupla) - Seleciona o elemento que tem um valor de chave imediatamente menor ao valor de chave corrente na variável Reltupla.

Adicionalmente o PASCAL/R oferece a função boolea na EOR(Rel) para indicar o fim da relação e a função SIZE(Rel) que indica o número de elementos na relação.

O exemplo a seguir ilustra o uso das primitivas e funções sobre relações:

```

BEGIN
    :
    VAR REL1,REL2: RELTIPO;
    :
    REL2 := [ ];
    LOW(REL1);
    WHILE NOT EOR(REL1) DO
        BEGIN
            :
            REL2 := REL1;
            NEXT(REL1);
        END;
    END;
END;

```

#### 5.9.5 Considerações sobre proteção dos dados

Nenhuma referência, sobre este assunto, foi constatada na bibliografia analisada.

#### 5.9.6 Estruturas de controle de fluxo

Para controle de fluxo, sobre relações, o PASCAL/R apresenta duas alternativas distintas:

. Através do uso do comando "FOR" com cláusula de seleção de relações.

. Através do uso dos comandos normais de iteração do PASCAL aliado à função EOR(rel).

##### a) Comando "FOR"

O comando FOR indica que um comando será executado repetidamente enquanto uma progressão de valores é trans

ferida para uma variável de controle.

Para este comando foi introduzida uma cláusula para a seleção de relações no seguinte formato:

```
FOR <seleção> DO <comando>
<seleção> ::= EACH <var.controle reg>
              IN <var.relação> : <expressão lógica>
```

Os valores das chaves dos componentes da relação na seleção não podem ser alterados pelo comando de repetição.

Ex.:

```
FOR EACH Reg IN Rel : TRUE DO
  BEGIN
    ;
  END
```

b) Comandos de iteração do PASCAL + função EOR

Para controle de fluxo sobre as relações o usuário pode, opcionalmente, utilizar os comandos de iteração ("WHILE DO" e "REPEAT UNTIL") do PASCAL em combinação com a função EOR.

Ex.:

```
WHILE NOT EOR(REL) DO
  BEGIN
    ;
    <comandos sobre tupla da relação>
  END
```

iteração enquanto a função EOR permanecer com o valor "falso".

O controle de fluxo pode também ser realizado através de um teste explícito sobre a função EOR.

### 5.9.7 Considerações finais

Desta análise da linguagem PASCAL/R pode-se concluir que, apesar de não se tratar especificamente de um sistema de gerência de banco de dados, com as novas construções implementadas como uma extensão da linguagem PASCAL, o sistema PASCAL/R adquiriu características que o deixam próximo da fronteira de uma linguagem de manipulação e gerência de dados armazenados em um banco de dados.

Convém ressaltar que o requisito faltante para que o PASCAL/R possa ser considerado um S.G.B.D é o aspecto relacionado com a proteção dos dados, principalmente no que se refere a um mecanismo para permitir acesso concorrente aos dados. Também um mecanismo de recuperação em caso de falhas seria de interesse, embora tenha alguns S.G.B.D., atualmente no mercado, que não possuem esta facilidade implementada.

Pode-se considerar que a linguagem PASCAL/R apresenta características que representam a materialização de objetivos do embutimento de linguagens de banco de dados em linguagens hospedeiras como:

- . Tratamento de todos os dados é feito de forma homogênea (definição de tipos e variáveis).
- . Instruções apresentam semelhanças com instruções PASCAL.
- . Facilidades de entrada/saída utilizadas igualmente.
- . Mesma estrutura do programa (com blocos) é utilizada com o banco de dados.

Em vista disso, a linguagem foi utilizada para comparação, embora seja considerada uma interface autocontida.

## 6. COMPARAÇÃO E CLASSIFICAÇÃO DAS CARACTERÍSTICAS DA DML NAS DIVERSAS FORMAS DE EMBUTIMENTO

### 6.1 Considerações preliminares

Com base na análise realizada sobre as linguagens de manipulação dos principais sistemas de gerência de banco de dados, é possível efetuar uma comparação e classificação das diversas formas de implementar o embutimento destas linguagens em linguagens de programação convencionais.

A comparação foi direcionada em 3 níveis distintos a fim de permitir maior clareza no exame das estruturas e construções utilizadas para proporcionar o acoplamento entre as duas linguagens. Tem-se, portanto a seguinte distribuição dos níveis de comparação:

- . comparação a nível de dados
- . comparação a nível de instruções
- . comparação a nível de mensagem.

Optou-se por esta distribuição em virtude da comunicação do usuário com o sistema de gerência de banco de dados ser claramente generalizada de forma que pode ser materializada, no programa de aplicação, através de três canais de comunicação. E é justamente estes três canais (dados, instruções, mensagens) que foram utilizados como alvo da comparação.

Para cada nível de comparação foram enfatizados todos os tópicos que se realçam e são representativos, comparativamente, nas diversas linguagens aqui exemplificadas. Peculiaridades de cada linguagem, separadamente, não foram tratadas aqui, visto que isto já foi estudado, anteriormente, no levantamento das características e na análise das diferentes linguagens de manipulação de dados.

A seguir será avaliado, com mais detalhes, cada

nível de comparação com os seus tópicos comparativos.

## 6.2 Comparação a nível de dados

Um dos componentes básicos de qualquer sistema de gerência de banco de dados é o instrumento utilizado para denominar e descrever os elementos de informação que compõem o banco de dados e para especificar os relacionamentos existentes entre estes elementos. No caso do embutimento das funções que realizam as operações sobre o banco de dados em uma linguagem hospedeira este instrumento deve ser, também, aplicado para possibilitar uma definição, a nível de programa de aplicação, dos dados constantes em um banco de dados. Muitas vezes este instrumento é muito versátil, sem grandes necessidades de especificações explícitas de layout e formato por parte do programador pois esta tarefa é desempenhada por facilidades de introdução automática de definição dos dados no fonte do programa, facilidades estas que são parte integrante do próprio projeto da interface. Outras vezes, no entanto, o programador precisa ter a preocupação de definir, explicitamente em seu programa, todos os dados que pretende extrair ou incluir em um banco de dados. Estas facilidades dependem da forma como o embutimento é realizado.

Foram enquadrados neste nível de comparação todos os aspectos relacionados com a definição dos dados que são, no âmbito do ambiente do programa de aplicação, manuseados pelos diversos comandos de manipulação (DML). Foram comparadas as diversas formas como o programador visualiza os dados contidos em um banco de dados, além das declarações que se fazem necessárias para que se materialize a visão local do banco de dados em um determinado programa de aplicação.

Recentemente facilidades adicionais de criação dinâmica de estruturas do banco de dados foram introduzidas



como características de algumas linguagens de manipulação de dados embutidas em linguagens hospedeiras. Este é um aspecto que também pode ser explorado na comparação.

Um terceiro aspecto, que também é considerado na comparação realizada a nível de dados, é o problema de conversões de tipos de dados do ambiente do S.G.B.D. para o ambiente do programa de aplicação e vice-versa.

#### 6.2.1 Como o usuário define a área de comunicação de dados

Conforme observado durante a análise das diferentes linguagens de manipulação de informações armazenadas em banco de dados, a forma como o canal de dados se materializa no ambiente do programa de aplicação, tanto para entrada como saída, é diferenciado para as diversas modalidades de embutimento.

Analisando detalhadamente a área de comunicação de dados em cada uma das linguagens, percebe-se peculiaridades em suas características que dificultariam uma comparação direta entre elas. Para melhor delinear esta comparação, pode-se focalizar a área de comunicação de dados questionando os seguintes pontos de vista:

- . Como o usuário descreve os dados desejados?
- . Como o usuário recebe os dados desejados?

Com base neste desmembramento do contexto, pode-se agora procurar uma comparação e classificação.

##### a) Como o usuário descreve os dados desejados

Sob este ponto de vista é considerada a maneira como se processa a materialização da descrição dos dados, referentes as estruturas do banco de dados, no programa de aplicação. Este aspecto está relacionado com o tipo de

preocupação que o usuário deverá ter para que obtenha, em seu programa, uma descrição dos dados necessários a sua aplicação.

Comparativamente, pode-se chegar à seguinte classificação:

- . A descrição é automaticamente introduzida no programa.

- . O usuário define, em seu programa, o layout completo da(s) estrutura(s) desejada(s).

- . O usuário define os campos de interesse de sua aplicação.

- . As estruturas e campos são naturalmente manipulados, a nível de programa, sem necessidade de definição.

Na primeira modalidade de definição, normalmente, figura o exemplo dos embutimentos implementados de forma compilativa, onde o compilador da linguagem hospedeira é modificado para reconhecer novos tipos e construções, ou nos embutimentos através do processo de precompilação. Nesta modalidade o programador recebe a descrição dos dados, que pretende manipular, introduzida no fonte de seu programa durante o processo de compilação. É uma modalidade que aparece no sistema DMS II e também no IDMS, ambos baseados no modelo DBTG da Codasyl. Para estes sistemas, o programador terá, como única tarefa de definição, especificar em pontos estratégicos do programa ("data-base section" para o DMS II e "schema section" para o IDMS) uma cláusula "DB" que indique ao compilador quais as estruturas que serão utilizadas e cuja descrição deverá ser trazida para dentro do programa.

Na segunda modalidade aparecem os embutimentos nos quais o programador deverá ter a preocupação de descrever todos os dados manualmente em seu programa de aplicação.

ção, considerando o registro completo de uma estrutura, ou então, opcionalmente, copiar a descrição de uma biblioteca de fontes previamente definida. De qualquer forma ele deverá ter conhecimento do layout dos dados que pretenderá acessar. Esta é uma modalidade de definição dos dados que aparece, geralmente, nos sistemas em que o embutimento da DML em linguagens hospedeiras é realizado através de chamada a subrotina como é o caso do IMS, onde, além da área de entrada e saída, o programador deverá também especificar um bloco de comunicação do programa (PCB) para que possa ser realizado o mapeamento entre o banco de dados lógico e o físico.

No caso do PASCAL/R, são encontradas características totalmente diferentes visto que não se trata de um embutimento de uma linguagem de manipulação em linguagem hospedeira e sim uma modificação de uma linguagem de programação convencional para operar, também, com construções do tipo relação e do tipo banco de dados. Nesta linguagem todo gerenciamento é por ela realizado e neste caso também a definição dos dados seguirá os padrões normais da linguagem. É somente mais um outro tipo de dado que pode ser definido no programa aproveitando, inclusive, tipos básicos existentes para formar as estruturas deste novo tipo.

De qualquer forma, abstraindo as suas características quanto ao tipo de linguagem e considerando que o usuário terá também, neste caso, a preocupação de descrever, em seu programa, todo o layout das estruturas, é possível incluir a linguagem PASCAL/R, igualmente, na segunda modalidade de classificação.

A modalidade seguinte na classificação que, pela abrangência contextual da descrição, constitui-se em uma variante da segunda modalidade, pode ser estabelecida para o caso dos sistemas TOTAL e ADABAS onde apesar do programador ter a incumbência de definir em seu programa as áreas de entrada e saída terá, também, a flexibilidade de especificar, através de parâmetros específicos na chamada a subro

tina, quais os campos que lhe interessam na aplicação. Com isto serão retornados somente os campos que realmente forem relevantes para o programa e que tenham sido requisitados. Aliado a isto, no sistema ADABAS aparece ainda a possibilidade de especificar o formato dos dados que serão manipulados no programa. Isto permite ao usuário a liberdade de especificar formatos adversos ao do formato de armazenamento no banco de dados sem ter a preocupação de pensar nas conversões, pois isto será providenciado pelo próprio ADABAS.

Uma outra modalidade, que aparece em sistemas onde o embutimento das linguagens de manipulação de dados (DML) em linguagens convencionais de programação é realizado por um processo de precompilação, pode ser constatada no SQL e EQUER (INGRES). Nesta modalidade não há propriamente uma definição explícita dos dados no programa. Os identificadores de domínios e relações aparecem naturalmente no programa sem que haja necessidade de descrevê-los no ambiente deste programa. Isto se deve ao fato do processo de precompilação reconhecer estes identificadores a partir de informações extraídas de arquivos descritores do sistema (catálogos) que guardam todo o contexto de representação das estruturas como nome das relações, nome dos domínios, formatos, tipos, etc. O usuário poderá, portanto, referenciar as relações e os domínios diretamente nos comandos de manipulação sem se preocupar em providenciar a definição destes dados.

#### b) Como o usuário recebe os dados desejados

Para completar o intercâmbio de comunicação entre o programa de aplicação e a interface do sistema de gerência de banco de dados, existe a necessidade de estabelecer as áreas de compartilhamento das informações onde dados são extraídos e entregues ao S.G.B.D. nas operações de atualização ou onde são recebidos pelo programa de aplicação nas operações de consulta. Este aspecto está vinculado ao

fato de haver a necessidade ou não de estabelecer, no programa, áreas adicionais para entrada/saída dos dados respectivamente recebidos ou enviados ao banco de dados.

Comparando as diversas linguagens analisadas chega-se à seguinte classificação:

. Entrada/saída feita sobre a mesma área da descrição dos dados.

. Entrada/saída em áreas declaradas para armazenar o conteúdo (concatenado) de campos solicitados.

. Entrada/saída sobre variáveis do programa.

A primeira modalidade caracteriza aquelas linguagens onde os dados oriundos do banco de dados são jogados sobre a área na qual está descrito todo o layout do registro, como no caso do IMS, DMS II, IDMS e PASCAL/R. Não existe, portanto, uma distinção entre a área que contém a descrição de formato e tipo dos diversos campos que compõem o registro e a área onde serão armazenados os conteúdos de cada campo. Esta modalidade aparece de forma típica no DMS II e IDMS onde todo processamento é realizado sobre a área de descrição do registro. O IMS também pode ser enquadrado nesta modalidade utilizando a área, onde o registro está descrito, como parâmetro no qual o conteúdo dos dados para entrada/saída é colocado. Também, no PASCAL/R, a entrada e saída de informações e todo o processamento intermediário é realizado sobre a mesma área onde as estruturas estão descritas pois, como já foi visto anteriormente, a definição de um banco de dados segue os padrões normais de definição e utilização dos tipos básicos da linguagem e nestes tipos básicos qualquer referência ao conteúdo da área é também feita através do nome com que foi declarada.

Na segunda modalidade figuram as linguagens onde a descrição dos dados é indicada em uma área diferente da-

quela onde é realizada a entrada e saída das informações. Nesta modalidade se enquadra o ADABAS e também o TOTAL. No ADABAS, por exemplo, a descrição é realizada numa área de formato onde é indicado o nome do campo de interesse e o seu formato e a recepção/transmissão dos dados é feita através da área do registro que terá o valor dos campos solicitados no formato indicado. Uma sistemática semelhante ocorre no caso do TOTAL onde uma seqüência de nomes de campos é colocada em um dos parâmetros da chamada a subrotina, porém sem nenhuma indicação de formato pois no TOTAL não é feita qualquer especificação deste tipo, e em um outro parâmetro aparece a área do usuário para entrada e saída das informações.

Nas linguagens SQL e EQUER (INGRES) aparece uma outra modalidade onde, como foi visto, nenhuma descrição das estruturas do banco de dados aparece ao nível do programa de aplicação. Nestas linguagens o usuário faz uso de variáveis da própria linguagem hospedeira nos comandos de manipulação, como variáveis intermediárias de recepção e transmissão de dados. Estas variáveis, no entanto, devem ser reconhecidas como tal pelo preprocessor, através de um sinal que as identifique. No SQL estas variáveis são precedidas por "\$" no comando e no EQUER precedidas por "++ ++" na sua declaração.

### 6.2.2 Criação dinâmica de estruturas do banco de dados a nível de programa

A capacidade de criação dinâmica de estruturas do banco de dados a nível de programa de aplicação é uma facilidade que proporciona um incremento substancial nas potencialidades de um sistema de gerência de banco de dados. Esta facilidade cria uma flexibilidade de não dependência exclusiva das estruturas geradas e autorizadas por um administrador do banco de dados. O usuário não fica vinculado,

estaticamente, ao repertório inicial de estruturas previstas no projeto do banco de dados. Com esta facilidade pode haver uma migração em direção a uma descentralização de funções e tarefas. Cada usuário estará apto a manter, sob sua responsabilidade, além das estruturas pertencentes ao banco de dados e cuja autorização de uso deverá possuir, um conjunto próprio de estruturas intermediárias de trabalho que ele terá a liberdade de criar e posteriormente destruir, se assim o desejar, além de, adicionalmente, poder realizar a concessão de autorização de uso destas estruturas para terceiros. Com isto pode-se conceber uma hierarquia nas liberações de autorizações onde no topo está o administrador do banco de dados e em subseqüentes níveis hierárquicos os diferentes usuários criando e autorizando, para o uso, as novas estruturas temporárias ou permanentes.

Por outro lado, isto poderá ir contra a filosofia de um banco de dados no que se refere ao uso disciplinado dos dados. Num banco de dados centralizados existe um maior controle sobre os dados que cada usuário utiliza.

Uma vantagem que pode ser realçada no uso de estruturas geradas dinamicamente pelo programa de aplicação reside no fato de que uma relação temporária poderia ser guardada para uso posterior no programa ou passada como parâmetro (ou referenciada) para outros programas. Indiretamente poderia ser reduzida a complexidade de uma consulta (consultas particionadas).

Uma classificação, quanto à criação dinâmica de estruturas pode ser realizada como segue:

- . Detém a facilidade de geração dinâmica de estruturas.
- . Não detém a facilidade de geração dinâmica de estruturas.

Dentre os sistemas analisados, encontra-se esta fa

cidade na linguagem SQL e também na linguagem EQU<sub>EL</sub> (INGRES) ambas enquadradas no modelo relacional. Como foi observado em nosso estudo e análise das linguagens de manipulação de dados, no EQU<sub>EL</sub> a criação de uma nova relação é possibilitada ao usuário através do uso do comando RETRIEVE INTO ou CREATE e estas relações são mantidas por um período de tempo especificado pelo administrador do banco de dados ou por um comando explícito SAVE. Já para a linguagem SQL, esta facilidade é conseguida pelo comando CREATE TABLE permitindo que cada usuário defina e crie componentes de seu banco de dados e tendo sobre eles o completo controle, sendo que uma carga de múltiplas tuplas sobre a nova tabela poderá ser realizada por um comando INSERT INTO combinado com um comando que selecione as tuplas (SELECT).

Realizando, novamente, a comparação com o PASCAL/R vê-se que a facilidade de criação de novas relações é conseguida através dos comandos de atribuição que foram adicionados como uma extensão do PASCAL original. Esta facilidade é porém uma característica natural do PASCAL/R pois, como já foi mencionado anteriormente, todo conjunto de estruturas do tipo relações e banco de dados é gerenciado pela própria linguagem como se fossem os tipos básicos convencionais manipulados pela linguagem. O contexto geral desta linguagem é diferente visto que se trata de uma linguagem autocontida mas o efeito conseguido com esta facilidade é o mesmo.

Já para os demais sistemas analisados não pode ser constatada esta característica pois todo processo de criação das estruturas do banco de dados é realizado estatisticamente pelo administrador do banco de dados que se vale de uma linguagem própria para esta finalidade e que é independente dos programas de aplicação que poderão somente manipular os dados armazenados nestas estruturas através dos comandos próprios para consulta e/ou atualização. Não existe disponível nenhum comando embutido na linguagem do programa de aplicação que permita realizar a criação de novas estru-



turas ou então eliminar as já existentes.

### 6.2.3 Conversões de tipos de dados

Quando se trabalha simultaneamente com dois ambientes diferentes de manuseio dos dados, ou seja, um ambiente onde os dados são gerenciados por um sistema de banco de dados e um outro ambiente onde os dados são processados em áreas de programas de aplicação, um problema típico de conversão dos dados compartilhados por estes dois ambientes pode vir a se caracterizar, caso estas linguagens forem acopladas.

Quanto a quem é atribuída a responsabilidade de realizar as conversões necessárias, pode ser considerado um assunto a ser enfatizado nas decisões do projeto. Em vista disto e uma vez analisadas as diversas linguagens de manipulação de dados acoplados a linguagens convencionais de programação é possível chegar-se a uma conclusão sobre a classificação que poderia ser dada ao aspecto de como são realizadas as conversões de tipos de dados. Esta classificação é:

- . Conversão automática a cargo do S.G.B.D.
- . Conversão baseada em indicação explícita de formato.
- . Nenhum procedimento de conversão é realizado por motivos diversos.

Dentre os sistemas estudados a "conversão automática a cargo do S.G.B.D." aparece no SQL e no EQUER. Para estes dois exemplos o sistema se encarrega de efetuar as conversões necessárias a fim de que os dados sejam passados de forma coerente para as variáveis do programa sem que haja a necessidade de uma intervenção do usuário. Também, pode ser considerada uma conversão automática nos sistemas IDMS e DMS II visto que uma descrição dos dados é automati

camente introduzida no fonte do programa e nos padrões da linguagem hospedeira.

Para o sistema ADABAS existe uma outra modalidade de conversão. Neste sistema o usuário tem a liberdade de especificar formatos adversos dos formatos em que os dados estão armazenados no banco de dados. Quando um programa de-seja utilizar os dados em outro formato, o usuário deverá informar explicitamente o novo formato para que o ADABAS faça as devidas conversões.

Nos outros sistemas como TOTAL, IMS e PASCAL/R, nenhum procedimento de conversão é realizado por motivos diversos. No TOTAL, o usuário receberá o conteúdo dos campos solicitados, concatenados em uma área contínua, sendo que cada campo ocupará o tamanho que lhe foi atribuído durante a definição e neste sistema não existe o conceito de tipo de dados diferentes pois todos os dados são tratados em termos de número de bytes. No IMS existe um procedimento semelhante apesar dos dados serem retornados em forma de registros e dentro do registro cada campo possuir o tamanho e formato originalmente atribuídos durante a definição dos dados. Nos dois exemplos, se o usuário desejar realizar alguma conversão, esta ficará sob sua inteira responsabilidade. E para o caso do PASCAL/R todas as operações que envolvem dados do tipo relação e do tipo banco de dados devem ser de mesmo tipo sendo considerado como uma restrição no uso das expressões manipuladas nos comandos da linguagem.

### 6.3 Comparação a nível de instrução

As instruções são igualmente um aspecto fundamental a ser considerado no confronto comparativo entre as diversas linguagens de manipulação de dados atualmente disponíveis no mercado. Trata-se de um aspecto que pode demonstrar claramente o nível de uma linguagem. A forma como as

instruções de manipulação dos dados são manuseadas, embutidas no ambiente de uma linguagem hospedeira, têm uma repercussão muito grande sobre a satisfação pessoal do usuário em delas fazer uso como ferramenta para atender as suas necessidades de manipular as informações armazenadas no banco de dados.

Nesta comparação procurou-se caracterizar os detalhes ligados à homogeneidade entre as linguagens acopladas que terá reflexos diretos sobre a conveniência, ou seja, sobre a adequação da linguagem resultante à satisfação das solicitações do usuário, ao poder de seleção das instruções, às estruturas de controle de fluxo necessárias e, também, aos aspectos de proteção visíveis a nível da interface de linguagem hospedeira.

Condicionado a estes aspectos foi traçado um paralelo entre os diversos sistemas de gerência de banco de dados analisados anteriormente.

### 6.3.1 Homogeneidade

A homogeneidade entre duas ou mais linguagens, quando estas são utilizadas simultaneamente por uma mesma pessoa, é um aspecto que merece especial atenção em uma análise comparativa de suas características pois, desta homogeneidade ou uniformidade, vai depender a maior ou menor satisfação do usuário em utilizar construções pertencentes a linguagens distintas (compreensão, aprendizagem e facilidade de comunicação) [CHA 80/].

A homogeneidade, do ponto de vista em que foi usada como um dos objetos da análise comparativa (verificando a uniformidade entre a linguagem de manipulação (DML) e a linguagem hospedeira), depende de uma certa forma, da maneira como foi implementado o embutimento e das características de cada DML comparadas com as da linguagem hospedeira.

Observando as características do embutimento de cada uma das linguagens de manipulação estudadas anteriormente pode-se focalizar a homogeneidade sob dois prismas distintos:

- . Homogeneidade da DML com a linguagem hospedeira alvo.

- . Homogeneidade entre os embutimentos da DML nas várias linguagens hospedeiras.

Uma vez feita a distinção dos critérios de verificação de homogeneidade em dois enfoques diferentes, resta agora realizar a comparação e classificação para cada um deles.

a) Homogeneidade da DML com a linguagem hospedeira alvo

Sob este prisma analisa-se a existência ou não de uma uniformidade entre as construções da DML comparadas com as construções de uma determinada linguagem hospedeira, utilizada como alvo, quando da descrição do S.G.B.D. Neste sentido pode ser questionado quais seriam as semelhanças existentes considerando, como universo de comparação, as construções da DML comparadas com as construções da linguagem hospedeira alvo.

Para este questionamento pode-se enquadrar a homogeneidade em uma das seguintes classificações:

- . Uniformidade sintática entre DML e comandos da linguagem hospedeira.

- . Uniformidade de tratamento de chamada a interface do S.G.B.D. com chamada a rotinas externas convencionais.

- . Padrões próprios da DML independentes dos padrões da linguagem hospedeira.

O primeiro caso pode ser exemplificado com o DMS II e o IDMS. Os projetos destes dois sistemas foram baseados no modelo proposto pelo DBTG da CODASYL [TAY 76/], que teve como um dos objetivos, padronizar uma linguagem de manipulação de banco de dados com uma linguagem convencional de programação (COBOL). Neste caso, como já foi visto anteriormente, as novas construções foram projetadas para que mantivessem o mesmo padrão sintático das demais construções da linguagem hospedeira. O programador utiliza as novas construções naturalmente como uma extensão da linguagem original sem que sejam necessárias grandes adaptações ou esforço de aprendizagem. Este tipo de homogeneidade pode também ser aplicada para o PASCAL/R.

O segundo caso aparece para os embutimentos realizados através de chamada a subrotina, como ocorre nos sistemas TOTAL, ADABAS e IMS. Nestes sistemas as instruções e demais informações de controle e áreas de comunicações entre os módulos são parametrizadas nas chamadas à interface do S.G.B.D. de forma análoga às chamadas de rotinas externas convencionais. É lógico que os parâmetros são codificados de acordo com os padrões de cada sistema ou rotina externa mas todas as chamadas são realizadas através de uma macro "CALL" que é uma das construções existentes na linguagem hospedeira e apresenta uma sintaxe própria em cada linguagem hospedeira e é independente da rotina externa que está sendo chamada.

Padrões próprios da DML independentes dos padrões da linguagem hospedeira é um exemplo típico que pode ser constatado no caso do SQL e EQUER. Neste caso os comandos de consulta e demais comandos de manipulação de dados utilizam os padrões próprios do sistema de gerência de banco de dados.

b) Homogeneidade entre os embutimentos da DML nas várias linguagens hospedeiras

Comparando as linguagens sob este prisma poderá ser caracterizado a existência de uniformidade confrontando as construções da DML com as construções em cada uma das linguagens hospedeiras. O universo de comparação passa agora a ser a visão em conjunto de todas as linguagens hospedeiras que comportam os comandos da DML. Será questionado agora se as construções DML embutidas em uma linguagem hospedeira têm os mesmos padrões das construções DML embutidas em uma outra linguagem hospedeira.

Neste sentido, pode-se ter a seguinte classificação:

- . Uniformidade das construções da DML para diversas linguagens hospedeiras.
- . Diferenciação conforme padrões da linguagem hospedeira.

Uma vez que, nas linguagens SQL e EQUERL são constatados padrões próprios para as construções DML independentes dos padrões da linguagem hospedeira, pode-se concluir que, para estas mesmas linguagens, existe uma uniformidade das construções da DML para diversas linguagens hospedeiras (mesmas construções da DML em linguagens hospedeiras diferentes).

Já no caso dos demais sistemas estudados, existe uma diferenciação conforme padrões da linguagem hospedeira. Para o DMS II e IDMS, embora o mnemônico de identificação de determinado comando DML seja idêntico nas diversas linguagens hospedeiras, existem diferenças sintáticas no uso do comando em uma ou outra linguagem hospedeira. Por mínimos que sejam, os detalhes sintáticos variam com a linguagem hospedeira. No TOTAL, ADABAS e IMS, onde a interface é

mantida através de chamada à subrotina, a diferença se observa na forma como a interface do S.G.B.D. é invocada em cada uma das linguagens hospedeiras e esta forma vai depender dos padrões sintáticos de cada linguagem hospedeira em particular.

### 6.3.2 Poder de seleção

O poder de seleção de uma linguagem de manipulação de dados (DML), quando esta se encontra embutida em uma linguagem convencional de programação, denota a habilidade que ela possui em expressar uma consulta cuja resposta está contida no banco de dados [/MIC 76/].

Pode-se afirmar que o poder de seleção de uma linguagem é um dos fatores que ditam o nível de sofisticação desta linguagem.

Com vistas em uma melhor distribuição das diversas linguagens de manipulação de dados estudadas, tentando classificá-las de acordo com o seu poder de seleção, a análise será conduzida sob dois pontos de vista diferentes:

- . Quanto ao protocolo de recuperação de registros
- . Quanto ao nível das consultas.

#### a) Quanto ao protocolo de recuperação de registros

Sob este ponto de vista, o objetivo da análise foi a verificação da potencialidade existente nos operadores da linguagem em termos de número de registros recuperados a cada instrução executada.

Em vista de, num ambiente de programa de aplicação, geralmente haver um tratamento individualizado de registros, as consultas DML embutidas nestes programas e que selecionarem diversos registros a cada instrução, logicamente

te deverão receber um tratamento diferenciado para sanar este impasse.

Por exemplo, as instruções de consulta da DML, ao invés de trazerem para as áreas de trabalho do programa todos os registros selecionados, elas simplesmente deixam assinalados os registros ou trazem referências destes registros ao ambiente do programa para que possam posteriormente serem recuperados por comandos adequados, ou, como outra alternativa, iterações explícitas codificadas em meio às instruções DML ficando, neste caso, o controle de fluxo sob responsabilidade desta instrução DML.

Considerando este detalhe na avaliação do protocolo de recuperação de registros, pode-se chegar à seguinte classificação:

- . Recuperação de registro único por instrução DML.
- . Assinalamento de um conjunto de registros por instrução DML.
- . Recuperação de um conjunto de registros por instrução DML.

Se enquadram no primeiro caso, ou seja, "Recuperação de registro único por instrução DML", os sistemas TOTAL, DMS II, IDMS, IMS e PASCAL/R. No DMS II e no IDMS, por exemplo, o comando "FIND" retorna exatamente um registro. Conseqüentemente, o usuário precisa executar a seqüência própria de comandos "FIND" para obter a informação completa. De forma análoga é realizada a recuperação dos registros no sistema TOTAL e IMS onde, através de uma instrução "CALL", é recuperada no parâmetro adequado desta chamada a subrotina (<área-ent-sai> no IMS e <registro> no TOTAL) a informação contida em um único registro (segmento no caso do IMS). O PASCAL/R, por sua vez, admite esta modalidade de recupera-



ção de registros através de suas primitivas de manipulação de relações (LOW, NEXT, THIS, ...) que retornam tuplas individuais a cada execução da instrução.

O "assinalamento de um conjunto de registros por instrução DML" pode ser exemplificado pelo SQL e também pelo ADABAS. No SQL, como foi visto, as instruções "SELECT" assinalam todas as tuplas que satisfazem o critério de seleção especificado. A partir deste assinalamento o usuário poderá trazer as tuplas, uma a uma, para o ambiente de seu programa através da instrução "FETCH" referenciando o cursor que foi associado à consulta. No ADABAS, por sua vez, esta característica se evidencia pela recuperação, com uma instrução "CALL", de uma lista de números internos de sequência (ISN) dos registros que satisfazem ao critério de pesquisa especificado. Com base nesta lista de ISN's o usuário estará em condições de recuperar cada registro que havia sido selecionado.

Como exemplo de "recuperação de um conjunto de registros por instrução DML" figuram o PASCAL/R, IMS e o EQUUEL. Para o PASCAL/R, por exemplo, existe a possibilidade de especificar instruções que atuam sobre todo um conjunto de tuplas de uma relação. Já no EQUUEL, a característica de recuperação de um conjunto de tuplas por instrução DML segue uma peculiaridade própria da linguagem onde é codificado um bloco de instruções da linguagem hospedeira, para processamento de cada tupla individual, acoplado à instrução da DML para formar a base de recuperação de todo o conjunto de tuplas da relação. A iteração de recuperação de cada tupla individual é implicitamente controlada pela instrução DML.

O IMS admite, opcionalmente, especificar posicionamento múltiplo codificando-se, no parâmetro relativo à lista de argumentos de pesquisa, um código de comando "\*D" através do qual é mantida a posição de cada segmento no caminho hierárquico de acesso permitindo, com isto, que registros de diferentes entidades (tipo de segmento) sobre um

mesmo pai sejam recuperados e processados em paralelo. Embora não admita a recuperação de múltiplos registros pertencentes a uma mesma entidade, o IMS pode também ser enquadrado na presente classificação.

b) Quanto ao nível das consultas

O nível de uma consulta pode ser expresso em termos de abrangência desta consulta, de granularidade na informação recuperada e de potencialidades existentes nas expressões de seleção.

b.1) Abrangência de uma consulta

O termo "abrangência de uma consulta" diz respeito à quantidade de conjuntos de dados que podem estar envolvidos em uma instrução de consulta ao banco de dados. A terminologia "conjuntos de dados", no conceito aqui considerado, assume diversos significados quando transposto para a denominação utilizada nos diversos sistemas de gerência de banco de dados. Assim tem-se, por exemplo, a denominação conjunto de dados (DATA-SET) para o TOTAL e DMS II, tipo de registro (RECORD TYPE) para o IDMS, tipo de segmento (SEGMENT TYPE) para o IMS, arquivo para o ADABAS e relações para o SQL, EQUER e PASCAL/R.

Sob o aspecto de abrangência de uma consulta, o nível de sofisticação desta consulta pode, portanto, ser classificado em:

- . Consulta sobre conjuntos de dados únicos
- . Consulta sobre múltiplos conjuntos de dados
- . Consulta sobre múltiplos conjuntos de dados porém previamente associados ou vinculados.

Para os sistemas TOTAL, DMS II e IDMS são permitidas somente consultas que envolvem em uma determinada instrução DML um único conjunto de dados. No TOTAL, por exem-

plo, existe um parâmetro específico na chamada a interface do banco de dados que armazena o nome do conjunto de dados alvo da consulta. Para o DMS II e também no IDMS é especificado, como uma das cláusulas do comando "FIND", o nome do conjunto de dados sobre o qual a consulta será realizada.

Consultas sobre múltiplos conjuntos de dados usando uma única instrução DML é uma característica que pode ser encontrada nos sistemas SQL, EQUER e também PASCAL/R. No EQUER esta facilidade se materializa usando simplesmente uma variável do tipo "tupla" (RANGE OF <var> IS <rel>) para cada relação que será usada na consulta e utilizar estas variáveis na instrução DML para qualificar os campos desejados. Para o PASCAL/R as relações desejadas em uma consulta são usadas diretamente nos comandos e expressões de seleção da DML. Para o SQL, por sua vez, se consegue a consulta sobre múltiplas relações através da associação entre as relações feitas com o uso do aninhamento das cláusulas de seleção.

O exemplo de uso de consultas sobre múltiplos conjuntos de dados, porém com uma associação ou vinculação previamente estabelecida, ocorre no caso dos sistemas ADABAS e IMS. Para o ADABAS pode-se especificar critérios de pesquisa, no parâmetro denominado "<área-da-condição>", que envolve até 5 (cinco) arquivos fisicamente acoplados sendo que o acoplamento é previamente determinado, por um comando "FILE <nro arq>(<campo>) WITH FILE..." durante a fase de definição dos dados. No caso do IMS as consultas obedecem a uma ordem de pesquisa pré-estabelecida dentro de um caminho hierárquico de acesso. O usuário pode especificar, como um dos parâmetros do comando "CALL", uma lista de argumentos de pesquisa utilizados para identificar os segmentos a serem recuperados e, neste caso, haverá um critério para cada tipo de segmento pertencente ao caminho de acesso.

## b.2) Granularidade na informação recuperada

Outro aspecto vinculado com o nível de consulta é a "granularidade com que uma informação pode ser recuperada". Este enfoque caracteriza uma consulta em termos de que dados podem ser recuperados, ou seja, a que nível de profundidade pode ser detalhada uma informação numa consulta ao banco de dados.

Para a granularidade na informação recuperada pode-se chegar à seguinte classificação:

- . recuperação a nível de registro
- . recuperação a nível de qualquer campo no registro
- . recuperação a nível de campos específicos.

Na recuperação a nível de registro, o registro é a unidade básica de recuperação das informações e aparece de forma típica nos sistemas DMS II, IDMS, IMS e PASCAL/R. Para o DMS II e para o IDMS cada instrução "FIND" trará para a área do programa os dados de um registro completo. O mesmo ocorre quando um comando "CALL" é executado no IMS com uma instrução DML de consulta adequada. No PASCAL/R, as operações podem, também, ser processadas com base nos dados de tuplas completas. No SQL, embora que em sua interface interativa permita indicar que todos os dados de uma tupla devam ser recuperados ("\*" substituindo lista de atributos no SELECT), quando embutido esta facilidade não pode ser constatada.

Nos sistemas ADABAS, TOTAL, SQL, EQUER e PASCAL/R o usuário tem a flexibilidade de especificar somente os campos que deseja utilizar em seu programa e pode ser qualquer campo dentro do registro. Neste caso são retornados somente os campos que foram solicitados.

No DMS II o usuário tem ainda a flexibilidade de recuperar somente o conteúdo de campos chaves de um determinado relacionamento (SET) ao invés de ser forçado a manipu-

lar o conteúdo de todo um registro mesmo não necessitando do restante das informações. Esta facilidade é conseguida através do uso do comando "FIND KEY OF" e é de grande utilidade nos procedimentos de validação de conteúdo de campos chaves para a atualização do banco de dados. Nestes casos o usuário realmente necessita saber somente o conteúdo das chaves que identificam os registros.

### b.3) Potencialidade existente nas expressões de seleção

Complementando a análise feita sobre o nível de uma consulta, pode-se considerar também, como característica que dita o grau de sofisticação de uma linguagem de manipulação de dados, a "potencialidade existente nas expressões de seleção" usadas como argumento de pesquisa sobre o banco de dados. Esta potencialidade expressa a habilidade que as instruções DML possuem para recuperar informações solicitadas pelos usuários.

Sob este aspecto, uma classificação da potencialidade encontrada nas expressões de seleção das diversas linguagens de manipulação, com seus respectivos embutimentos em linguagens hospedeiras, pode seguir os critérios abaixo:

- . seleção com base em valor de qualquer campo
- . seleção com base no valor de determinado campo (chave de acesso)
- . seleção com base em identificação interna de registro
- . seleção com base em caminhos pré-estabelecidos.

Para enquadramento das características das instruções DML, como sendo do tipo que seguem uma "seleção com base em valor de qualquer campo", não foram considerados os aspectos ligados ao desempenho da consulta, resultante de uma pesquisa sobre um valor de campo não declarado como chave primária ou secundária. Foram considerados como sendo en

quadráveis nesta modalidade de seleção todas as instruções DML que, de uma forma ou outra, permitem fazer a recuperação de uma determinada informação quando é conhecido o valor de qualquer um dos campos que estão associados a esta informação. Podem ser enquadrados nesta modalidade os sistemas SQL, EQUER, PASCAL/R, ADABAS e IMS. Para o SQL e o EQUER as expressões de seleção aparecem especificadas na cláusula "WHERE" dos comandos de recuperação, respectivamente "SELECT" e "RETRIEVE". No PASCAL/R o critério de seleção de um determinado valor de campo aparece, como pode ser visto pela BNF da linguagem, especificado em uma expressão lógica que qualifica a tupla desejada. No ADABAS, por sua vez, o valor de qualquer campo pode ser testado na consulta, especificando-se o critério de seleção num parâmetro conhecido como "área-de-condição" e valor do campo a ser usado na comparação num parâmetro denominado "área-do-valor". Para o IMS os critérios de seleção aparecem na lista de argumentos de pesquisa especificado como um dos parâmetros da chamada a interface do S.G.B.D.

Existem, no entanto, sistemas que permitem somente uma "seleção com base no valor de determinado campo" geralmente definido como chave de acesso ao registro. Isto ocorre no sistema TOTAL onde todas as operações não seriais exigem a especificação de uma chave primária do registro em um dos parâmetros da chamada a interface. No caso do DMS II, quando o usuário desejar realizar uma consulta que exija o teste do conteúdo de determinado campo, deverá utilizar o comando "FIND" com a cláusula "AT", e este campo de consulta deve estar declarado no SET/SUBSET como chave ou como item de dado incorporado a esta estrutura auxiliar.

A "seleção com base em identificação interna de registro" pode ser encontrada nos sistemas IDMS, ADABAS e TOTAL. No IDMS, o usuário especifica um item de dado que conterá a chave de acesso do banco de dados (DATABASE-KEY) usando o comando "FIND <reg> DB-KEY IS <id>". No ADABAS, também, o usuário tem a possibilidade de acessar uma ocorrência de registro pelo seu número interno de seqüência (ISN)

usando o comando "READ IN", por exemplo. O TOTAL admite também a possibilidade de acessar um registro pelo seu ponteiro de referência interno (IRP) caso o programador tenha salvo, anteriormente, este endereço relativo, para poder usá-lo em posteriores operações de recuperação direta de registros membros.

A "seleção com base em caminhos pré-estabelecidos" é, geralmente, utilizada nos casos em que se deseja recuperar os dados em determinada ordem previamente estabelecida (operações seriais). Para tal pode haver a necessidade de uso de estruturas auxiliares do sistema. O uso de estruturas auxiliares aparece por exemplo no sistema DMS II que processa os dados em certa seqüência lógica especificada com auxílio de estruturas do tipo "SET", "SUBSETS" e "ACCESS". No IDMS existe, também, a possibilidade de especificar uma ordenação nos relacionamentos (denominados "SETS" no IDMS) entre entidades owner e member. Neste sistema o usuário pode especificar explicitamente que deseja recuperar os dados através destes relacionamentos (FIND NEXT <reg> WITHIN <set>). Para os sistemas DMS II e IDMS os registros podem ser processados sobre estas estruturas auxiliares segundo uma seqüência navegacional, ou seja, selecionar o primeiro (FIRST), o último (LAST), o próximo (NEXT) e o anterior (PRIOR).

A característica de seleção com base em caminhos pré-estabelecidos pode, também, ser constatada nas operações seriais encontradas em alguns sistemas, como o IMS e TOTAL. No IMS, existe uma ordenação implícita para selecionar os registros na seqüência hierárquica pré-estabelecida e o acesso é feito com uma instrução do tipo "GET NEXT" (GN). No TOTAL, por sua vez, a pesquisa aos registros membros pode ser realizada seguindo o caminho estabelecido pela ligação entre estes registros. No PASCAL/R embora possua primitivas de tratamento de relações do tipo tupla com valor imediatamente maior (NEXT) ou imediatamente menor (PRIOR), não se enquadra nesta classificação por não haver uma ordenação prévia das tuplas e conseqüente caminho de acesso pré-estabelecido.

### 6.3.3 Estruturas de controle de fluxo

O tipo de estrutura de controle de fluxo lógico disponível, em apoio às instruções da DML, para recuperação dos dados nas operações seriais, é um dos fatores que ditam a sofisticação e complexidade de uma linguagem de programação que tenha embutido, em seu ambiente, construções de uma linguagem de manipulação de dados.

Quanto mais poderosa a atuação das construções que manipulam as iterações, mais estruturado e conciso será o programa de aplicação propiciando, com isto, uma maior legibilidade na forma de programar.

É importante que se deixe frisado que o controle explícito de fluxo lógico do programa é uma característica que induz uma proceduralidade na linguagem, visto que desta forma o programador estará indicando, passo-a-passo, todos os procedimentos necessários à recuperação dos dados a serem retornados do banco de dados. Pode-se afirmar, portanto, que as estruturas de controle de fluxo são, também, determinantes da proceduralidade ou não da linguagem.

Convém esclarecer, no entanto, que o termo "proceduralidade de uma consulta" tem provocado consideráveis discussões. Uma consulta não procedural relata meramente o que é o resultado da consulta e não como é obtido. Uma consulta procedural, por sua vez, descreve o processo para alcançar a meta. Esta classificação, no entanto, não é discreta e a idéia não é claramente definida [REI 81/]. Linguagens orientadas a conjuntos, como SQL e EQUER, são usualmente consideradas como sendo não procedurais. Outras linguagens, como DMS II e IDMS, já se aproximam mais de características procedurais. Por outro lado, traçando um paralelo entre SQL, DMS II e ADABAS, por exemplo, o conceito de proceduralidade já pode trazer certas divergências.

Analisando as características dos embutimentos, pa



ra os diversos sistemas de gerência de banco de dados estudados, foi possível chegar-se à seguinte classificação das estruturas de controle de fluxo:

- . controle de iteração a nível da linguagem hospedeira
- . controle de iteração a nível da DML.

Quando o protocolo de recuperação dos registros do banco de dados for do tipo "um-registro-por-vez", ou seja, a cada instrução DML é recuperado somente um registro, como no caso dos sistemas TOTAL, IMS, IDMS e DMS II, presume-se que, logicamente, as iterações tenham que ser codificadas pelo programador através de carga explícita dos registros, com o controle do fluxo se completando mediante um teste do código de estado retornado ao programa de aplicação pela interface do S.G.B.D., após concluída a instrução DML. Também nas linguagens SQL e ADABAS, cuja consulta faz o assinalamento de um conjunto de registros, existe a necessidade de especificar um controle de fluxo explícito com teste de código de estado quando desejar-se trazer cada registro para o ambiente do programa de aplicação. No SQL as tuplas (registros) são individualmente trazidas para o programa através do comando "FETCH" atuando sobre um cursor. No ADABAS, uma vez assinalados os registros por um comando "FIND NORMAL" (S1), a transferência de cada um, para a área do programa, é realizada por um comando "READ ISN" (L1). O PASCAL/R, por sua vez, pode, opcionalmente, valer-se dos comandos normais de iteração da linguagem PASCAL, aliado a função EOR(REL), para materializar uma estrutura de controle de fluxo explícita.

A segunda modalidade de controle de iteração, ou seja, controle de fluxo a cargo da DML, aparece no EQUERL e também no PASCAL/R. Ambas as linguagens recuperam um conjunto de tuplas a cada instrução DML. Como foi visto, no EQUERL, para processamento de todas as tuplas que satisfazem os critérios de seleção em uma consulta, haverá um bloco de ins-

truções da linguagem hospedeira acoplado à instrução da DML, mas o controle do fluxo lógico é mantido por esta instrução DML. Já no PASCAL/R existe, também, a modalidade onde as tuplas podem ser recuperadas individualmente sob o controle de iteração realizado pelo comando "FOR EACH ...", que foi adaptado do comando original "FOR" do PASCAL, especificamente para atuar sobre as relações. Existe, portanto, uma analogia com as instruções DML do EQUCEL.

#### 6.3.4 Proteção de dados

A proteção de dados integra, também, os aspectos vinculados com a segurança e integridade e, neste sentido, pode-se considerar que as implicações oriundas do compartilhamento de um banco de dados por diversos usuários faz da proteção dos dados armazenados uma função essencial do sistema de gerência de banco de dados. Esta proteção deverá, portanto, atuar sobre duas formas distintas: proteção na qualidade; proteção na segurança.

Proteção na qualidade implica na imunidade do banco de dados contra alterações ou exclusões inválidas, enquanto proteção na segurança implica na imunidade do banco de dados contra alterações ou exclusões não autorizadas.

A proteção de um banco de dados compartilhado pode ser conseguida através da especificação de diversas restrições de integridade e restrições de segurança. Estas restrições direcionam o sistema de gerência de banco de dados a aplicar as verificações apropriadas para garantir que o programador esteja autorizado a executar as operações especificadas e, quando da alteração do banco de dados, para garantir que todas modificações são válidas e que a consistência do banco de dados é mantida.

No caso do embutimento de algumas linguagens de manipulação de banco de dados em linguagem de programação

convencional, o programador deve ter a preocupação de especificar certas cláusulas ou comandos que indiquem, ao sistema, que uma proteção sobre os dados deva ser estabelecida. Em outros casos isto já está implícito no sistema de gerência de banco de dados.

Dentre os assuntos de interesse da análise comparativa das linguagens, foram consideradas, aqui, 3 formas de proporcionar uma proteção dos dados:

- . quanto à atualização concorrente (proteção propriamente dita)
- . quanto à atualização inválida (integridade)
- . quanto à atualização não autorizada (segurança e privacidade).

a) Quanto à atualização concorrente

O problema da proteção dos dados, quanto à atualização concorrente, surge quando diversos usuários procuram realizar, simultaneamente, modificações no banco de dados ou quando certos usuários necessitam obter informações deste banco de dados enquanto este estiver sendo alterado.

Sob este ponto de vista, um enquadramento das características das diversas linguagens de manipulação de banco de dados estudadas pode focalizar a seguinte classificação:

- . chaveamento implícito ao sistema
- . chaveamento especificado na definição dos dados
- . especificação explícita de comandos DML de chaveamento e/ou modo de operação
- . nenhum controle está implementado.

Um "chaveamento implícito ao sistema" aparece no EQUER e no SQL. No EQUER é mantido um chaveamento implícito em cada comando de atualização e a nível de relação, sendo que a informação sobre quais relações estão sendo bloquea-

das em dado momento, é mantida em relações específicas para esta finalidade (LOCK RELATION). Para o SQL, também, existe um mecanismo de chaveamento automático dos objetos manipulados por comandos de atualização (bloqueio físico: páginas; bloqueio lógico: relações, segmentos, identificadores de tuplas, ...).

"Chaveamento especificado na definição dos dados" pode ser vista nos sistemas IDMS e DMS II. O IDMS tem a possibilidade de contornar problemas causados por atualizações concorrentes especificando uma maneira apropriada do uso das áreas lógicas destinadas ao "SUBSCHEMA" através de um comando do tipo "PRIVACY LOCK FOR EXCLUSIVE UPDATE IS YES". Com este comando é assegurado o acesso exclusivo durante a atualização da área em questão. O DMS II, por sua vez, admite a especificação de bloqueio para os data-sets embutidos através do comando "LOCK TO MODIFY DETAILS", sendo que o efeito deste comando é o de assegurar que nenhum acesso seja feito ao data-set disjunto (pai) enquanto estiver sendo alterado o data-set embutido (filho) vinculado a este.

A "especificação explícita de comandos DML de chaveamento e/ou modo de operação" aparece para a maioria dos sistemas como TOTAL, ADABAS, IMS, IDMS, DMS II e SQL. No TOTAL os problemas de atualização simultânea podem ser evitados através das funções "RESERVE" e "RELEASE" que garantem a exclusividade ou não no acesso a arquivos (a função "RESERVE" indica o chaveamento setando um "LOCK BYTE"). Existe também a possibilidade de especificar o modo de operação (somente leitura; atualização), informado no comando SIGN-ON [/DAV 77/]. No ADABAS, igualmente, existe a possibilidade de especificar o modo de operação e, aliado a isto, a existência de comandos com opção de bloqueio ou chaveamento como, por exemplo, "READ HOLD". Para o IMS existe chaveamento de segmentos recuperados quando for especificado comandos do tipo "GHU", "GHN" ou "GHNP". Para o IDMS pode ser especificado um modo de operação com o comando "READY" quan

do da inicialização das áreas relativas ao programa. No DMS II aparece o comando "LOCK/MODIFY" para proporcionar o chaveamento. Já no caso do SQL, como foi visto anteriormente, existe um chaveamento implícito a cada comando. Entretanto, uma seqüência de comandos pode ser condensada em operações atômicas (não interrompíveis) utilizando-se transações especificadas pelos comandos "BEGIN-TRANSACTION" e "END-TRANSACTION". Com isto, estes comandos estão, indiretamente, controlando o bloqueio para um conjunto de operações sobre o banco de dados.

Analisando o sistema PASCAL/R, pode-se verificar que ele se enquadra no caso onde "nenhum controle está implementado".

b) Quanto à atualização inválida

Focalizando o caso da proteção dos dados quanto a atualização inválida, cabe salientar o fato de que estas atualizações inválidas do banco de dados podem resultar de alterações realizadas de forma inadvertida, imprópria ou ainda maliciosa. Uma forma em que a integridade pode ser prejudicada é através de um erro do usuário que produz um resultado de atualização inesperado.

Sob o aspecto de atualizações inválidas pode-se enquadrar as diversas linguagens de manipulação de banco de dados estudadas na seguinte classificação:

- . controle realizado pelo S.G.B.D.
- . controle exclusivamente programado pelo usuário.

De uma maneira geral, pode-se considerar que a maioria dos S.G.B.D., atualmente disponíveis no mercado, tem implementado, como facilidade, algum controle quanto à atualização inválida. Este controle pode ser mais ou menos apurado dependendo da potencialidade do sistema. Assim, é possível enquadrar no primeiro caso os sistemas TOTAL, ADABAS,

IMS, IDMS, DMS II, EQUER e SQL. No TOTAL não existe um controle sofisticado. Os controles que aparecem neste sistema dizem respeito, por exemplo, à verificação de inserção de registro com chave primária em branco ou duplicada e a verificação da exclusão de registro mestre que ainda possua registros membros vinculados. Já no ADABAS existe um controle mais aprimorado visto que os dados são submetidos a uma verificação de formato assim que passarem pela área do registro e caso ocorra distorção nestes formatos um erro é notificado ao programa de aplicação. O IMS possui características muito limitadas e deficitárias para controle de atualização inválidas. Realiza, no entanto, uma verificação da existência do campo chave no segmento raiz, e permite a especificação de algumas restrições de acesso (E - exclusão, R - alteração, I - inclusão) quando da definição do bloco de comunicação do programa (PCB). No IDMS existe a possibilidade de especificar a não duplicação de chaves (DUPLICATES ARE NOT ALLOWED) e realizar uma certa proteção dos dados contra exclusões indevidas através de um comando "PRIVACY LOCKS" especificado a nível de registro. Para o DMS II existem controles mais sofisticados permitindo a especificação de cláusulas como "REQUIRED" (preenchimento obrigatório de campo), "VERIFY" (verificação de uma condição), "NO DUPLICATES" (não duplicação de valores) e "READONLY" (somente leitura permitida). No EQUER, tem-se a possibilidade de especificar asserções de integridade através do comando "INTEGRITY CONSTRAINT" e no SQL aparecem controles para atualizações inválidas mediante o uso do comando "ASSERT" aliado a um mecanismo de definição de gatilho (TRIGGER's) para forçar a execução de certos comandos quando um evento ocorrer.

No caso do PASCAL/R, no entanto, como foi visto durante a análise da linguagem, nenhum controle é realizado para assegurar que não haja armazenamento de informações incoerentes ou inválidas e que prejudiquem a integridade do sistema. Para evitar que atualizações impróprias sejam realizadas, o usuário deverá realizar o controle através de

seu programa de aplicação (consistência prévia dos dados e batimento com informações constantes no banco de dados).

c) Quanto à atualização não autorizada

Outro aspecto a ser considerado na proteção dos dados diz respeito à atualização não autorizada. Percebe-se que uma brecha da segurança de um banco de dados é normalmente o resultado de alguns usuários ganharem acesso a dados para os quais não têm direito. Acesso a informação sensível, tal como salários, dados pessoais, saldo bancário devem ser restritos através de algumas formas de autorizações de acesso. E é sobre este aspecto que pode ser feita uma avaliação das características das diversas linguagens de manipulação de banco de dados estudadas e desta avaliação extrair uma classificação no seguinte critério:

- . nenhuma facilidade implementada para controle de acesso
- . autorização explícita atribuída ao usuário
- . autorização implícita nas visões locais estabelecidas para o programa.

No primeiro caso podem ser enquadrados os sistemas TOTAL e PASCAL/R visto que, nestes sistemas, nenhum controle de acesso está implementado.

Na segunda modalidade poderiam ser enquadrados os sistemas ADABAS, EQUER e SQL. Este controle de autorização pode ser estabelecido sobre um programa específico ou sobre todos os acessos ao banco de dados realizados por um determinado usuário. O ADABAS está dotado de facilidades de especificação de controle de acesso a nível de programa, arquivos ou campos. Para isto, cada programa deve fornecer, por ocasião da abertura do banco de dados, uma senha no bloco de controle, associado ao qual existe um nível de segurança. Com isto o programa só terá acesso aos campos que tiverem nível de segurança inferior ao estabelecido para este pro-

grama. No EQUER e SQL existe a facilidade de fornecimento dinâmico de autorização de acesso a determinados dados ou relações sem a necessidade de intervenção direta do administrador do banco de dados (EQUER: comando "RESTRICT"; SQL: comando "GRANT"). Com isto um usuário poderá passar a autorização de acesso a outros usuários, desde que ele tenha permissão para isto, e uma verificação de autorização para cada programa do usuário é realizada quando do acesso ao banco de dados.

A facilidade de definição de vistas aparece nos sistemas ADABAS (macros definidas pelo ADAMINT), IMS (definição de PSB a partir de PCB's), IDMS (definição de SUBSCHEMA), DMS II (REMAPS e BD lógico), EQUER (através do comando "DEFINE") e SQL (através do comando "DEFINE VIEW"). Para estes sistemas, uma vez possuindo a facilidade de definição de vistas, existe a possibilidade de manter o controle de acesso mediante a autorização implícita nas visões locais estabelecidas para o programa.

#### 6.4 Comparação a nível de mensagens

Para que o intercâmbio entre o programa do usuário e o sistema de gerência de banco de dados seja mantido, deverão figurar neste cenário, além das requisições feitas à interface através do canal de instruções e o compartilhamento dos dados mediante um canal para esta finalidade, também um canal onde o sistema informe ao usuário, através de mensagens adequadas, eventos sobre o estado da operação realizada sobre o banco de dados. Neste canal retornam as informações sobre o sucesso ou não da operação executada além de outras mensagens que sejam do interesse do usuário para que este fique a par de todas as ocorrências constatadas durante o acesso ao banco de dados.

Percebe-se, portanto, que as mensagens retornadas



ao programa de aplicação podem tanto indicar uma situação de sucesso na operação como uma situação de exceção encontrada durante a execução. No entanto, para generalizar a manipulação de mensagens, normalmente a situação de sucesso na operação é um teste adicional incorporado aos procedimentos para tratamento de exceções. Em vista disso, a comparação a nível de mensagens, para as diversas linguagens de manipulação de banco de dados estudadas, será realizada considerando-se os aspectos estruturais ligados ao tratamento de exceções em cada uma das linguagens.

#### 6.4.1 Tratamento de exceções

A cada operação sobre o banco de dados, requisitando dados ou enviando-os para que sejam armazenados, o sistema de gerência de banco de dados retornará ao programa de aplicação, através de sua interface, uma mensagem indicando o sucesso na execução da instrução fornecida ou acusando o tipo de falha constatada. Uma vez retornada, a mensagem deverá receber um tratamento adequado para que, a partir dela, seja determinado um fluxo de procedimentos que manipule o evento produzido, pelo acesso ao banco de dados, de acordo com a natureza de seu efeito. O tipo de procedimento para tratamento da mensagem vai depender de cada situação ou também da repercussão que a mensagem tem sobre os dados manipulados pelo programa.

A intenção aqui, no entanto, não é a de analisar o tipo de procedimento a ser especificado para que as mensagens sejam devidamente tratadas, mas sim verificar as facilidades que as linguagens de manipulação de banco de dados oferecem para sinalizar as mensagens ao programa de aplicação e a influência que estas facilidades têm sobre o tratamento de mensagens que de princípio serão consideradas como "exceções".

Analizando as diversas linguagens estudadas sobre

o aspecto de tratamento de exceções e comparando as suas facilidades neste sentido, pode-se chegar a uma classificação, para avaliar o tipo de construções disponíveis na linguagem que auxiliem na manipulação das mensagens. Verifica-se, portanto, que o tratamento de exceções pode ser ativado através das seguintes modalidades:

- . através de cláusulas específicas para sinalizar exceções
- . através de testes explícitos sobre o conteúdo da mensagem
- . através de manipulação automática por rotinas do S.G.B.D.

Cláusulas específicas para sinalizar exceções aparecem como integrantes de construções DML do DMS II e SQL. No DMS II, o teste do conteúdo do registrador "DMSTATUS" normalmente é usado em combinação com uma cláusula "ON EXCEPTION" ou com uma cláusula "USE ON DMERROR" a fim de permitir a especificação de procedimentos adequados. No SQL existe, opcionalmente, a "WHENEVER" para manipulação de exceções.

Testes explícitos sobre o conteúdo da mensagem é a opção para a maioria das linguagens de manipulação de banco de dados embutidas em linguagens convencionais de programação. Aparece, por exemplo, no TOTAL, ADABAS, IMS, DMS II, IDMS, SQL, EQUER e PASCAL/R. Em algumas linguagens como TOTAL, ADABAS e IMS, as mensagens são retornadas em parâmetros da chamada a interface do S.G.B.D. Em outras, como IDMS, DMS II, SQL e EQUER, o código de condição aparece em variáveis específicas para esta finalidade. Já no PASCAL/R existe, por exemplo, uma função booleana "EOR(Rel)" para indicar quando a relação está com um estado de fim da relação.

Adicionalmente, em alguns sistemas, o tratamento de exceções pode ser manipulado também, automaticamente, por rotinas do próprio S.G.B.D. Aparece, por exemplo, no IDMS

onde durante a definição dos dados pode ser especificada a associação de uma rotina de tratamento de exceções a determinadas operações sobre a referida estrutura que está sendo definida. Também no INGRES (EQUEL) existem facilidades para manipulação automática de exceções por meio de uma rotina padrão do sistema ou então por uma rotina especificada pelo usuário.

#### 6.5 Quadro sintético de comparação das características

Uma vez realizada a comparação e também a classificação das características das linguagens de manipulação de dados, nas diversas formas de embutimento destas em linguagens convencionais de programação, a forma ideal de ilustrar um sumário de todas as conclusões estabelecidas desta análise é a de mostrar as características num "quadro sintético de comparação", baseado nos padrões fixados no decorrer da análise comparativa.

			TOTAL	ADABAS	IMS	IDMS	DMS-II	EQUEL	SQL	PASCAL R	
INTERFACE DE LING. HOSPEDEIRA	IMPLEMENTAÇÃO DO EMBUTIMENTO	CHAMADA À SUBROTINA									
		PRECOMPILAÇÃO DE INSTRUÇÕES DML									
		EXTENSÃO DO COMPILADOR DA LINGUAGEM HOSPEDEIRA									
INTERFACE AUTOCONTIDA											
<b>COMPARAÇÃO A NÍVEL DE DADOS</b>											
COMO O USUÁRIO DEFINE A ÁREA DE COMUNICAÇÃO DE DADOS	COMO O USUÁRIO DESCRIBE OS DADOS DESEJADOS	DESCRIÇÃO AUTOMATICAMENTE INTRODUZIDA NO PROGRAMA									
		O USUÁRIO DEFINE O LAYOUT COMPLETO DAS ESTRUTURAS DESEJADAS									
		O USUÁRIO DEFINE OS CAMPOS DE INTERESSE DE SUA APLICAÇÃO									
	COMO O USUÁRIO RECEBE OS DADOS DESEJADOS	ESTRUTURAS E CAMPOS SÃO NATURALMENTE MANIPULADAS SEM NECESSÁRIA DEFINIÇÃO									
		ENTRADA/SAÍDA FEITA SOBRE A MESMA ÁREA DA DESCRIÇÃO DOS DADOS									
CRIAÇÃO DINÂMICA DE ESTRUTURAS DO B. D. DO PROGRAMA	DETÉM A FACILIDADE DE GERAÇÃO DINÂMICA DE ESTRUTURAS										
	NÃO DETÉM A FACILIDADE DE GERAÇÃO DINÂMICA DE ESTRUTURAS										
CONVERSÕES DE TIPOS DE DADOS	CONVERSÃO AUTOMÁTICA A CARGO DO S.G.B.D.										
	CONVERSÃO BASEADA EM INDICAÇÃO EXPLÍCITA DE FORMATO										
	NENHUM PROCEDIMENTO DE CONVERSÃO É REALIZADO POR MOTIVOS DIVERSOS										
<b>COMPARAÇÃO A NÍVEL DE INSTRUÇÕES</b>											
HOMOGENEIDADE	HOMOGENEIDADE DA DML COM A LINGUAGEM HOSPEDEIRA ALVO	UNIFORMIDADE SINTÁTICA ENTRE DML E COMANDOS DA LINGUAGEM HOSPEDEIRA									
		UNIFORMIDADE DE TRATAMENTO DE CHAMADA A INTERFACE S.G.B.D. COM CHAMADA A ROTINAS EXT. CONVENCIONAIS									
		PADRÕES PRÓPRIOS DA DML INDEPENDENTES DOS PADRÕES DA LINGUAGEM HOSPEDEIRA									
HOMOGENEIDADE ENTRE OS EMBUTIMENTOS DA DML NAS VÁRIAS LING. HOSP.	HOMOGENEIDADE ENTRE OS EMBUTIMENTOS DA DML NAS VÁRIAS LING. HOSP.	UNIFORMIDADE DAS CONSTRUÇÕES DA DML PARA DIVERSAS LINGUAGENS HOSPEDEIRAS									
		DIFERENCIAÇÃO CONFORME PADRÕES DA LINGUAGEM HOSPEDEIRA									
		QUANTO AO PROTOCOLO DE RECUPERAÇÃO DE REGISTROS									
PODER DE SELEÇÃO	QUANTO AO PROTOCOLO DE RECUPERAÇÃO DE REGISTROS	RECUPERAÇÃO DE REGISTRO ÚNICO POR INSTRUÇÃO DML									
		ASSINALAMENTO DE UM CONJUNTO DE REGISTROS POR INSTRUÇÃO DML									
		RECUPERAÇÃO DE UM CONJUNTO DE REGISTROS POR INSTRUÇÃO DML									
	QUANTO AO NÍVEL DAS CONSULTAS	ABRANGÊNCIA DE UMA CONSULTA	CONSULTA SOBRE CONJUNTO DE DADOS ÚNICO								
			CONSULTA SOBRE MÚLTIPLOS CONJUNTOS DE DADOS								
			CONSULTA SOBRE MÚLTIPLOS CONJUNTOS DE DADOS, PORÉM PREVIAM. ASSOCIADOS								
		GRANULARIDADE NA INFORMAÇÃO RECUPERADA	RECUPERAÇÃO A NÍVEL DE REGISTRO								
			RECUPERAÇÃO A NÍVEL DE QUALQUER CAMPO NO REGISTRO								
POTENCIALIDADE EXISTENTES NAS EXPRESSÕES DE SELEÇÃO	RECUPERAÇÃO A NÍVEL DE CAMPOS ESPECÍFICOS										
	SELEÇÃO COM BASE NO VALOR DE QUALQUER CAMPO										
	SELEÇÃO COM BASE NO VALOR DE DETERMINADO CAMPO										
ESTRUTURAS DE CONTROLE DE FLUXO	CONTROLE DE ITERAÇÃO A NÍVEL DA LINGUAGEM HOSPEDEIRA	SELEÇÃO COM BASE EM IDENTIFICAÇÃO INTERNA DO REGISTRO									
		SELEÇÃO COM BASE EM CAMINHOS DE ACESSO PRÉ-ESTABELECIDOS									
PROTEÇÃO DOS DADOS	QUANTO A ATUALIZAÇÃO CONCORRENTE	CONTROLE DE ITERAÇÃO A NÍVEL DA DML									
		CHAVEAMENTO IMPLÍCITO AO SISTEMA									
		CHAVEAMENTO ESPECIFICADO NA DEFINIÇÃO DOS DADOS									
		ESPEC. EXPLÍCITA DE COMANDOS DML DE CHAVEAMENTO									
	QUANTO A ATUALIZAÇÃO INVÁLIDA	NENHUM CONTROLE ESTÁ IMPLEMENTADO									
		CONTROLE REALIZADO PELO S.G.B.D.									
		CONTROLE EXCLUSIVAMENTE PROGRAMADO PELO USUÁRIO									
	QUANTO A ATUALIZAÇÃO NÃO AUTORIZADA	NENHUMA FACILIDADE IMPLEMENTADA P/CONTR. ACESSO									
		AUTORIZAÇÃO EXPLÍCITA ATRIBUÍDA AO USUÁRIO									
		AUTORIZAÇÃO IMPLÍCITA NAS VISÕES LOCAIS DO PROG.									
<b>COMPARAÇÃO A NÍVEL DE MENSAGENS</b>											
TRATAMENTO DE EXCEÇÕES	ATRAVÉS DE CLÁUSULAS ESPECÍFICAS PARA SINALIZAR EXCEÇÕES										
	ATRAVÉS DE TESTES EXPLÍCITOS SOBRE O CONTEÚDO DE MENSAGENS										
	ATRAVÉS DE MANIPULAÇÃO AUTOMÁTICA POR ROTINAS DO S.G.B.D.										

Figura 6.1 - Quadro sintético de comparação de características

## 6.6 Conclusões delineadas sobre a comparação realizada

A partir do quadro sintético de comparação, apresentado na seção anterior, e da comparação de características das DML's, realizada neste capítulo, podem ser extraídas algumas informações que identifiquem os reflexos que as características da linguagem de manipulação dos dados tem sobre a maneira como o embutimento foi implementado e a influência da abordagem de estruturação dos dados (hierárquico, em rede, relacional) sobre este mesmo embutimento.

Algumas das conclusões, extraídas da comparação e classificação realizadas, aparecem relatadas a seguir.

. Focalizando o aspecto da definição da área de comunicação de dados, pode-se chegar a um parecer de que, geralmente, a forma como o usuário descreve os dados no programa fica na dependência da forma de como está implementado o embutimento. Para o embutimento por chamada a subrotina não existe, por exemplo, uma descrição da base de dados, introduzida automaticamente no programa de aplicação, por intermédio de uma simples indicação de uma diretiva para esta finalidade, codificada no referido programa. Para que esta diretiva possa ser utilizada no programa, existe a necessidade de uma modificação do compilador ou um pré-processor, a fim de que este a reconheça. Esta adaptação do compilador, certamente, não segue os princípios a que se propõe o embutimento por chamada a subrotina, que tende a oferecer uma forma simplificada de interface com o S.G.B.D. No caso do embutimento por chamada a subrotina, a descrição dos dados, relativos à base de dados, fica ao encargo do próprio usuário (pode opcionalmente copiar de uma biblioteca previamente definida) que deverá, em certas situações, definir o layout completo das estruturas desejadas ou somente os campos de interesse de sua aplicação. A característica de introduzir automaticamente a descrição da base de dados no programa de aplicação, mediante o uso de uma direti-

va, aparece nos embutimentos do tipo extensão do compilador, como no DMS II, ou eventualmente, nos embutimentos do tipo precompilação (caso do IDMS) em vista da própria sistemática que o projeto de embutimento, com esta característica implementada, se propôs a seguir, ou seja, proporcionar uma extensão natural das construções da linguagem hospedeira. Apesar disso, para os embutimentos do tipo precompilação implementados em algumas linguagens de manipulação de banco de dados de abordagem relacional, como SQL e EQUER, parece a manipulação das estruturas e campos sem necessidade de uma definição ao nível do programa de aplicação. Percebe-se, portanto, que nos embutimentos por precompilação existe mais de uma modalidade de como o usuário descreve a área de comunicação de dados. Como fatores que levam à escolha de uma ou outra modalidade pode ser mencionada a existência ou não de homogeneidade entre construções da DML e da linguagem hospedeira, que pode facilitar a tarefa de introduzir a descrição no programa de aplicação ou a necessidade de possuir ou não, disponível no ambiente do programa de aplicação, a descrição completa ou parcial da base de dados a fim de facilitar a manipulação de registros completos (depende da granularidade na informação recuperada em cada S.G.B.D.). A precompilação, portanto, não impõe a introdução automática da descrição na área do programa podendo também figurar os casos onde não haja necessidade de definição da base de dados com as estruturas e campos sendo, naturalmente, utilizadas pelas instruções DML (caso do SQL e EQUER).

. Com respeito a área de recebimento dos dados, no ambiente do programa, não há um critério de classificação rígido que segue um padrão dependente da forma de implementar o embutimento. O que se percebe, no entanto, é que, dentre os sistemas analisados, existe um certo consenso nos sistemas que seguem a abordagem relacional, com embutimento realizado pelo processo de precompilação, em efetuar a entrada e saída dos dados sobre variáveis do programa. Isto, provavelmente, se deve ao fato da abordagem relacional pres

supor a recuperação a nível de qualquer campo e de várias relações. Logicamente, haveria também a possibilidade, neste caso, de adotar o método utilizado pelos sistemas TOTAL e ADABAS, que também possuem uma recuperação a nível de campo mas onde, porém, a entrada/saída é realizada em áreas declaradas para armazenar o conteúdo concatenado dos campos solicitados. No entanto, realizando a entrada/saída sobre variáveis do programa, a tarefa de manipulação dos dados pode ser facilitada, visto que não existe a necessidade de especificar máscaras sobre a área de dados para ter acesso a campos individualmente, e o processo de conversão automática aparece simplificado pois as rotinas de conversão não terão, nestes casos, a preocupação de avaliar diferentes situações de conversões sobre uma mesma área de dados, área esta representada por um conjunto de campos cujo conteúdo vem concatenado.

. No que tange a criação dinâmica de estruturas do banco de dados, observa-se que esta é uma característica que aparece nos sistemas de abordagem relacional. Esta facilidade ficaria muito difícil de implementar nos outros sistemas, principalmente porque, nos sistemas que seguem a abordagem hierárquica e em redes, a definição da base de dados é realizada por uma interface específica para esta finalidade, inteiramente independente dos programas de aplicação, além desta definição valer-se de estruturas de dados mais complexas, com detalhamento de parâmetros físicos e conseqüente independência de dados mais restrita. No caso da abordagem relacional, os dados são estruturados utilizando um único tipo de construção, ou seja, através de relações. Já nos sistemas de abordagem em rede e hierárquicos, os dados aparecem estruturados através de diversos tipos de construções diferentes, como por exemplo, relacionamentos entre campos demarcados dentro de registros, estruturas auxiliares (sets, apontadores, caminhos de acesso) e outras construções que dependem de sistema em particular. Além disso, outro problema, que poderia surgir da criação dinâmica

de estruturas nos sistemas de abordagem hierárquica ou em redes, é o dos efeitos colaterais a nível lógico, ou seja, a implicação que teria, por exemplo, a criação de um novo SET que fosse utilizar registros já definidos anteriormente.

. Com respeito à taxonomia para conversões, pode-se ressaltar que existe uma certa dependência da forma como está implementado o embutimento, se bem que as peculiaridades dos tipos de dados do S.G.B.D. comparadas com as peculiaridades dos tipos que, normalmente, são encontrados nos programas de aplicação detêm, também, um certo grau de influência (existência de compatibilidade evita um mecanismo de conversão). Verifica-se, por exemplo, que nos casos dos embutimentos implementados mediante extensão do compilador, a descrição da base de dados é automaticamente introduzida no fonte do programa de aplicação e nos padrões da linguagem hospedeira. Nestes casos, deve haver uma conversão automática para adequar os tipos aos padrões da linguagem hospedeira e isto deve ser transparente ao usuário. A mesma premissa pode ser aplicada aos embutimentos realizados por precompilação, nos casos onde o usuário utiliza as estruturas e campos, naturalmente no programa de aplicação, sem a necessidade de declará-las. Nesses casos, mesmo não necessitando declarar as estruturas da base de dados no ambiente do programa, haverá a necessidade de uma conversão automática para o caso onde as instruções DML atuam concomitantemente, também, sobre as variáveis do programa de aplicação. No aspecto de conversões não pode ser atribuída uma dependência da abordagem de estruturação dos dados.

. A forma como é implementado o embutimento pode ditar uma classificação na homogeneidade da DML com a linguagem hospedeira. Para o caso do embutimento por chamada a subrotina haverá uma uniformidade de tratamento de chamada a interface do S.G.B.D. com a chamada a rotinas externas convencionais. Por outro lado, para que se possa implementar o embutimento através de uma extensão do compilador, de



verá haver uma uniformidade sintática entre a DML e os comandos da linguagem hospedeira. Esta uniformidade, no entanto, não é uma peculiaridade somente dos embutimentos realizados desta forma. Pode aparecer, também, nos casos de precompilação (IDMS, por exemplo). A implantação do embutimento por extensão do compilador fica, portanto, na dependência de uma homogeneidade sintática entre DML e comandos da linguagem hospedeira. No entanto a recíproca não é verdadeira. Igualmente, padrões próprios da DML, independentes dos padrões da linguagem hospedeira, são implementados através de um processo de precompilação mas isto não implica que, com o processo de precompilação, ocorra sempre esta situação.

. A homogeneidade entre os embutimentos da DML nas várias linguagens hospedeiras vai ser, também, um fator determinante da escolha da forma de embutimento pois, se for desejada uma uniformidade das construções DML para as diversas linguagens hospedeiras, a forma mais aconselhável de realizar o embutimento é através da precompilação.

. O protocolo de recuperação de registros, conduzindo a um manuseio de conjunto de registros, simultaneamente, em uma única instrução DML, fica condicionado a existência de construções que comportem a recepção deste conjunto de registros. Esta preocupação é parte integrante do projeto de um S.G.B.D. Quando, no entanto, esta facilidade é transposta para o ambiente do programa de aplicação, formas alternativas, para transferir os registros para a área do programa, devem figurar. Por exemplo, assinalar os registros selecionados para depois recuperá-los individualmente por uma instrução DML específica (caso do SQL e ADABAS) ou então proporcionar comandos da DML que mantenham, sob seu controle, o fluxo de liberação dos registros ao programa (caso do EQUER). A influência que esta característica teria sobre a forma de implementar o embutimento reside no fato de que, nos casos de chamada a subrotina, onde as áreas de en-

trada e saída devem ser, explicitamente, definidas e descritas, o usuário deverá prever a recuperação de mais de um registro por instrução (ex.: concatenação de registros no caso do IMS) especificando um tamanho de área coerente com o número de registros recuperados. Já no embutimento sendo por precompilação, onde as instruções utilizam as construções do próprio S.G.B.D., sem necessidade de declará-las no programa, e com a existência de estruturas de controle de fluxo adequadas, esta preocupação de previsão de recuperação de mais de um registro pode se tornar transparente, visto que a própria instrução DML poderá realizar um controle de iteração para liberar cada registro ao programa.

. A abrangência de uma consulta pode, de certa forma, depender da abordagem de estruturação dos dados pois, nas linguagens relacionais, aparece, como característica, a facilidade de realizar consultas sobre múltiplos conjuntos de dados. Nas abordagens hierárquicas ou em rede, o acesso a múltiplos conjuntos de dados já figura de forma mais restrita visto que fica na dependência de uma associação prévia entre estes conjuntos de dados. Os reflexos que esta característica possui sobre a escolha da forma de implementar o embutimento reside, basicamente, na maneira como o usuário deverá definir a área de comunicação de dados. Se a definição desta área ficar ao encargo do usuário, ele deverá ter a preocupação de prever o recebimento de dados contidos em diversos conjuntos de dados. A consulta sobre múltiplos conjuntos de dados tem, no entanto, a sua maior influência na disciplina de programação, pois, caso o S.G.B.D. não admita o acesso a múltiplos conjuntos de dados por instrução, o usuário, para obter o mesmo efeito, deverá programar toda uma rotina para acesso e recuperação das informações contidas nos diversos conjuntos de dados.

. Quanto à granularidade na informação recuperada, não pode ser constatada nenhuma dependência direta da forma de embutimento. Não se pode afirmar, também, que haja uma exclusividade na classificação da granularidade na in-

formação recuperada dependendo da abordagem de estruturação dos dados. Mesmo que a abordagem relacional determina uma granularidade por atributos isto não impede que sistemas de outras abordagens também tenham esta característica. Para o aspecto de granularidade na informação recuperada pode-se, no entanto, observar que geralmente existe uma vinculação com a maneira como o usuário define a área de comunicação de dados. Quando, por exemplo, a recuperação for a nível de qualquer campo no registro, os dados são retornados em variáveis do programa (SQL, EQUOL) ou em áreas especificamente declaradas para receber o conteúdo concatenado dos campos solicitados (TOTAL, ADABAS). Quando, porém, a recuperação for a nível de registro, estes serão retornados sobre a própria área de sua descrição.

. Também com respeito a potencialidade existente nas expressões de seleção, pelo critério de classificação adotado na taxonomia descrita, não existe uma dependência direta da forma de embutimento.

. No que se refere a estruturas de controle de fluxo, pode-se considerar que o controle de iteração a nível da DML é uma facilidade que pode ser implementada somente nos embutimentos do tipo precompilação ou ainda do tipo extensão do compilador. Para os embutimentos por chamada a subrotina a iteração deve ser controlada a nível da linguagem hospedeira.

. A forma de conciliar um tratamento de exceções está diretamente vinculada com a forma como o embutimento é implementado. Nos embutimentos realizados de forma precompilativa ou extensão do compilador, facilmente poderá haver procedimentos para tratamento de exceções associadas a uma cláusula específica para sinalizar os eventos. O mesmo já não ocorre para os embutimentos por chamada a subrotina.

Para sintetizar estas conclusões, é possível tecer alguns comentários sobre o que se pode generalizar para

os S.G.B.D.s que usam a mesma abordagem:

- Os sistemas de abordagem relacional, quando a DML é acoplada a uma linguagem de programação convencional, adotam a implementação do embutimento através da precompilação pois, geralmente, procuram utilizar padrões próprios da DML independentes dos padrões da linguagem hospedeira e uma uniformidade das construções da DML nas diversas linguagens hospedeiras sendo que estas características são obtidas através do processo de precompilação. Como justificativa destes sistemas procurarem seguir estas características poderia ser considerado o fato de que a abordagem relacional utiliza construções e facilidades que normalmente não aparecem como integrantes de uma linguagem de programação (linguagens de programação normalmente não admitem operações de mais alto nível, como álgebra relacional).

- Relacionado com o comentário anterior, pode-se afirmar que, com a abordagem relacional, os S.G.B.D., geralmente, procuram realizar a entrada e saída de informações sobre variáveis do programa de aplicação e as estruturas e campos da base de dados são naturalmente manipulados sem a necessidade de definição ao nível do programa. Em consequência, deverá haver uma conversão automática de tipos de dados a cargo do S.G.B.D. caso os tipos não sejam compatíveis.

- A capacidade de geração dinâmica de estruturas é uma característica dos sistemas de abordagem relacional pelos motivos expostos anteriormente nesta seção.

- A abordagem relacional já pressupõe, no protocolo de recuperação de registros, uma manipulação de conjunto de registros por instrução DML; na abrangência de uma consulta, um acesso sobre múltiplos conjuntos de dados por instrução DML; na granularidade da informação, uma recuperação a nível de qualquer campo no registro; e na potencialidade existente nas expressões de seleção, uma seleção com base no valor de qualquer campo.

- Com respeito às abordagens hierárquicas e em rede é possível afirmar que, atualmente, não possuem implementado a característica de criação dinâmica de estruturas.

## 7. ESTUDO DE CASO: ENQUADRAMENTO DE LOBAN NA CLASSIFICAÇÃO PROJETADA

### 7.1 Considerações preliminares

Com a análise realizada sobre as mais expressivas linguagens de manipulação de banco de dados em cada abordagem de estruturação de dados, a fim de realizar um levantamento de suas principais peculiaridades funcionais e operacionais, foi possível elaborar-se uma comparação e classificação das características de cada linguagem de manipulação de dados, considerando todos os aspectos relacionados direta ou indiretamente, com o embutimento destas nas linguagens convencionais de programação.

Concluindo este estudo, que culminou com a confecção do quadro sintético das comparações e conclusões a seu respeito, tenciona-se, agora, realizar uma avaliação prática sobre a taxonomia oriunda deste trabalho. Para tanto, foi escolhido o protótipo da linguagem LOBAN sendo implementado na Universidade Federal do Rio Grande do Sul, através do Curso de Pós-Graduação em Ciência da Computação, para ser utilizada como instrumento de referência para esta verificação. Esta avaliação visa realizar os refinamentos e ajustes necessários sobre os itens abordados na classificação e, simultaneamente, permitir o enquadramento da linguagem LOBAN no contexto geral de linguagens de banco de dados utilizadas nas interfaces de linguagens hospedeiras.

Inicialmente será realizada uma descrição de LOBAN nos moldes utilizados para descrever os diversos S.G.B.D. analisados nesta dissertação; em seguida será efetuada a avaliação de LOBAN no contexto da classificação projetada.

## 7.2 Descrição de LOBAN no contexto da classificação

### 7.2.1 Caracterização do sistema

#### 7.2.1.1 Estruturas de dados

A estruturação dos dados no SISTEMA L [/GOL 81/, /GOL 82/, /HEU 79/, /HEU 81/, /HEU 83/] relembra uma abordagem relacional. As informações são organizadas em arquivos relacionais que juntos constituem a base de dados acessível ao usuário.

Um arquivo relacional é composto por uma TABELA RELACIONAL (TR) e, opcionalmente, uma tupla sob nome FICHA que contém informações globais sobre a tabela relacional.

Os componentes do arquivo relacional podem ser manipulados por diversos operadores incluindo, também, os da álgebra relacional aplicados sobre tabelas relacionais.

O sistema oferece, também, outro tipo de tabela chamada TABELA LIGACIONAL (TL) que pode ser obtida como resultado da execução de operações, mas não aparece na base de dados. Esta tabela é uma coleção de ligações, onde cada ligação representa a associação entre uma tupla ligante e uma tabela relacional (tabela de ligados). Desta forma, a ligação é comparável a uma ocorrência de SET da abordagem CODASYL/DBTG.

#### 7.2.1.2 Linguagens hospedeiras

A linguagem de comunicação do usuário com o SGBD é conhecida como LOBAN (Linguagem de Operação de BANCO de Dados) [/SAN 80/].

O protótipo de implementação do embutimento da linguagem LOBAN em uma linguagem de programação convencio-

nal foi projetado considerando, como linguagem hospedeira, a linguagem PASCAL [GOL 82/].

Com o embutimento de LOBAN em PASCAL, o programa de aplicação escrito na linguagem oriunda deste acoplamento passou a receber a denominação de programa PLOBAN.

### 7.2.1.3 Estrutura funcional

A arquitetura geral do SISTEMA L pode ser vista na figura 7.1.

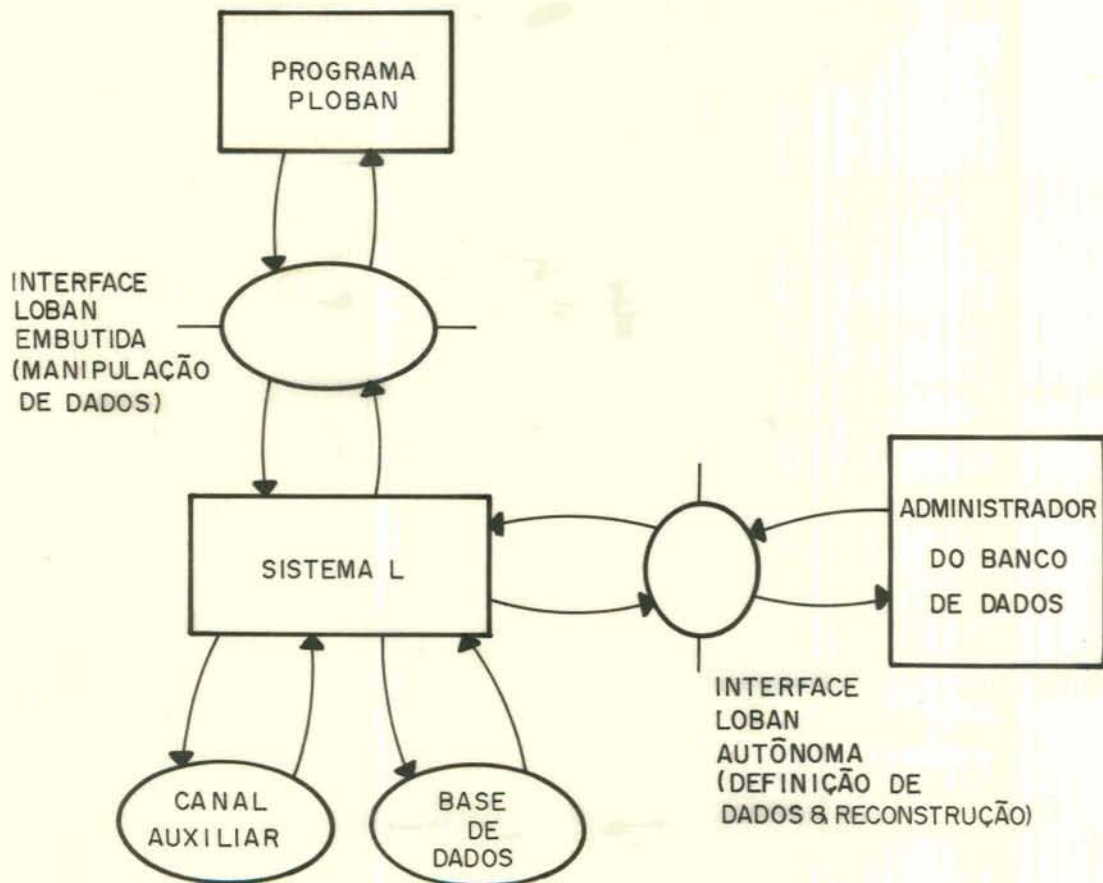


Figura 7.1 - Arquitetura funcional do SISTEMA L



## 7.2.2 Interface de definição dos dados no sistema

### 7.2.2.1 Definição da base de dados

Para a definição das construções na base de dados, são utilizados os conceitos de TIPO de construção.

Um TIPO de construção é um conjunto de construções que possuem os mesmos predicados. Todas as construções armazenadas na base de dados devem ser ocorrências de tipos de construções, sejam tipos definidos pelo usuário ou tipos genéricos, previamente definidos (PRETIPOS).

Na definição dos tipos o usuário fornece uma designação para o tipo, o pretipo ao qual pertence e os predicados do tipo.

A definição de agregados é feita através da especificação da COMPOSIÇÃO (especificação dos tipos dos componentes imediatos) e da CONEXÃO (especificação de relações entre os componentes de cada construção do tipo em definição).

Exemplo:

```
TABELA-EMP: TIPO DE TAREL
            TAL QUE
            COMPOS TUPLA-EMP
            CONEX CHAVE-PRIMARIA EMP-NRO
```

```
TUPLA-EMP: TIPO DE TUPLA
            TAL QUE
            COMPOS
                EMP-NRO    → TIPO-NUMERO
                EMP-NOME   → TIPO-NOME
                EMP-DATAADM→ TIPO-DATA
                EMP-IDADE  → TIPO-IDADE
                EMP-ENDER  → TIPO-ALPHA
```

EMP-GER → TIPO-NUMERO  
EMP-FUNCAO → TIPO-FUNCAO  
EMP-LOTACAO → TIPO-DEPTO

#### 7.2.2.2 Definição de esquemas externos

Na atual versão do sistema não existe implementada esta facilidade.

#### 7.2.3 Aspectos funcionais do embutimento

##### 7.2.3.1 Comunicação usuário com o banco de dados

Os canais de comunicação do programa de aplicação com o sistema de gerência de banco de dados aparecem ilustrados na figura 7.2.

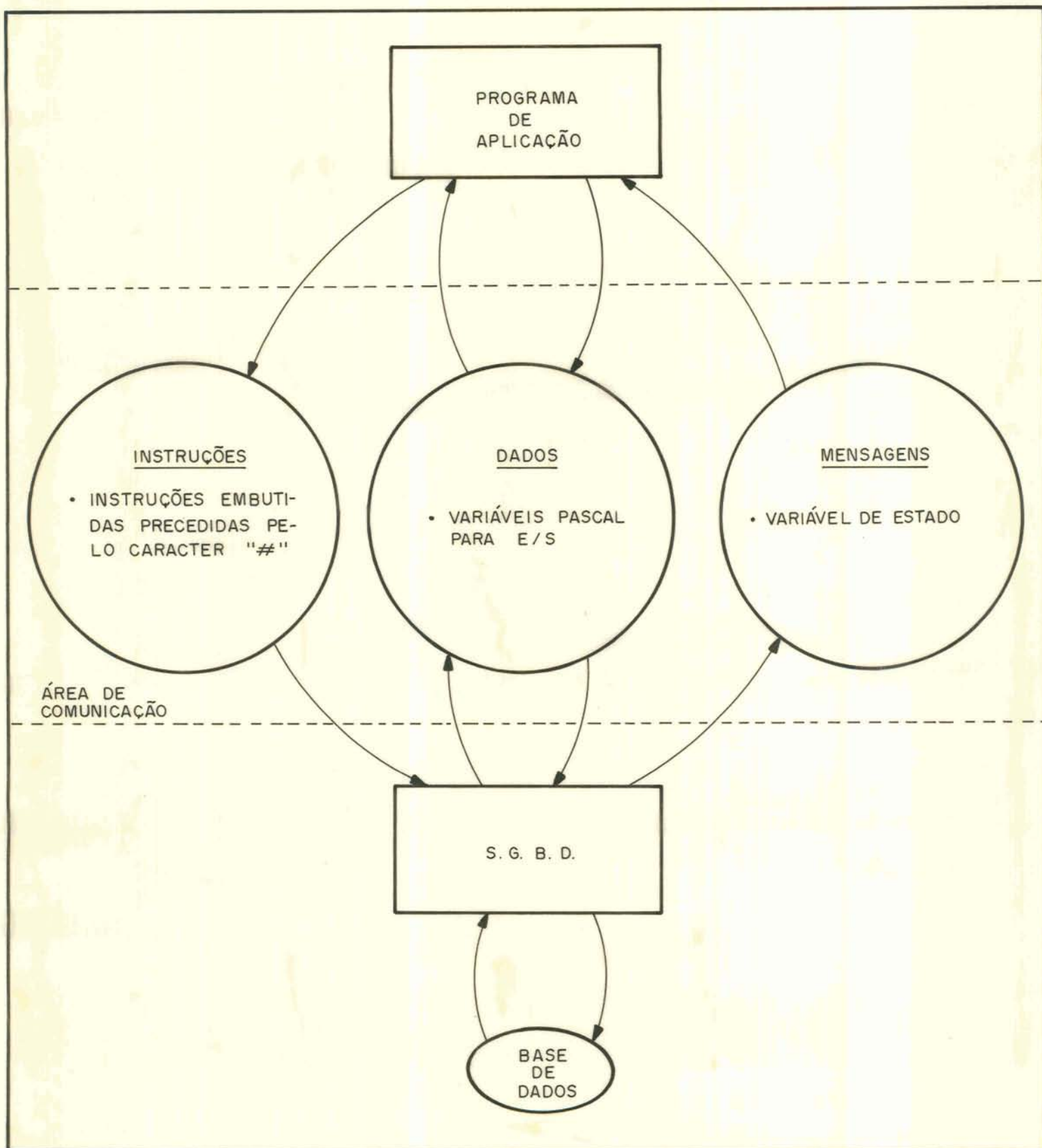


Figura 7.2 - Canais de comunicação programas de aplicação e SISTEMA L

## a) DADOS

Num programa escrito em PLOBAN todas as construções da base de dados definidas anteriormente podem ser referenciadas e manipuladas naturalmente sem a necessidade de declará-las no ambiente deste programa de aplicação. Isto se deve ao fato do processo de precompilação do fonte reconhecer as como tal.

Portanto, todas as consultas poderão ser realizadas sem haver a necessidade de antes declarar as construções que serão manipuladas.

No entanto, para a entrada e saída de dados da base de dados serão utilizadas variáveis declaradas em PASCAL. Estas variáveis poderão receber construções atômicas (array de caracteres, inteiros, reais) da base de dados ou canal auxiliar, através da instrução REPRESENTAR, ou poderão fornecer dados em uma instrução, através do operador INT (INTERPRETAR).

Exemplo:

```

:
VAR EMPREGADONOME: ARRAY [1..30] OF CHAR;
PROCEDURE PRINTNAME(NOME:ARRAY[1..30] OF CHAR);
:
†† REPRESENTAR C EMP-NOME EM EMPREGADONOME
   PRINTNAME (EMPREGADONOME)

```

O sistema permite, também, ao usuário a opção de especificar dinamicamente em seu programa de aplicação construções temporárias que serão mantidas até o fim da execução deste programa em um CANAL AUXILIAR. Os tipos das construções a serem mantidas no canal auxiliar devem ser tipos existentes ou definidos no programa através da instrução USAR VERBETES DE COERENCIA.

A referência às construções, em LOBAN, é feita através de uma forma, conhecida como ENDEREÇO DE PONTOS, por intermédio da qual uma construção é localizada em determinado contexto. A referência é feita da seguinte forma:

```
CRITERIO[. CRITERIO]...
```

onde cada critério serve para seleção de pontos em um nível de agregação de construções. O critério pode ser o nome da construção no ponto a selecionar ou uma expressão booleana complexa, que deverá ser verdadeira para que o ponto seja selecionado.

Exemplos:

```
ARQEMP.TR
ARQEMP.FICHA
```

b) INSTRUÇÕES

As instruções LOBAN embutidas no programa fonte PASCAL são precedidas por um sinal cardinal ("‡"). Com isto elas são reconhecidas como tal pelo precompilador.

Exemplo:

```

:
:
VAR EMPREGADONOME: ARRAY [1..30] OF CHAR,
:
:
PROCEDURE PRINTNAME(NOME:ARRAY [1..30] OF CHAR);
:
:
‡ FAZER PARA CADA TUPLA
‡     EM ESTREITAR
‡         (COLEC ARQEMP.TR.
‡             (C EMP-NRO='ØØ8')
‡         )

```

```

#          PARA EMP-NOME
# ((
# REPRESENTAR C EMP-NOME EM EMPREGADONOME
#   PRINTNAME (EMPREGADONOME);
# ))
:

```

Para que uma construção referenciada por um endereço de ponto seja obtida, LOBAN possui, associado ao critério de seleção de ponto, um operador denominado C. Este operador traz para a memória uma cópia da construção que está sendo referenciada. A obtenção das construções endereçadas é feita, portanto, através da aplicação do operador C sobre o endereço do ponto especificado.

Existem outros operadores que trabalham sobre endereços de pontos como o COLEC que seleciona um conjunto de tuplas que satisfazem o critério de seleção e o CARD que contabiliza o número de tuplas selecionadas.

Além desses, existem outros operadores que podem ser usados em uma <expressão de obtenção de construções> e que permitem a manipulação das construções obtidas. Dentre os principais, podem ser enumerados os seguintes:

- . ESTREITAR - PARA : estreitamento da tabela (projecção)
- . JUNTAR - COM - POR: junção de tabelas
- . DESAGRUPAR : obtenção de uma tabela relacional a partir de uma ligacional (concatenação de tuplas ligadas e ligantes)
- . LIGAR - COM - POR : obtenção de uma tabela ligacional a partir de duas relacionais
- . AGRUPAR - POR : obtenção de uma tabela ligacional pelo agrupamento de

- tuplas de uma tabela relacional (semelhante ao GROUP BY do SQL [/IBM 83/])
- . COMPOR : obtenção de tupla ou ligação a partir de construções do B.D., canal auxiliar ou variáveis PASCAL
  - . INT : operação de entrada de dados no sistema a partir de variáveis PASCAL
  - . SOMA/MEDIA/MAX/MIN/DESV - SOBRE : funções matemático/estatísticas

É nas expressões de obtenção de construções que reside a potencialidade das instruções de LOBAN, sendo utilizadas na manipulação dos dados e também na DEFINIÇÃO. Os operadores têm, entre outras, funções do mesmo nível das da álgebra relacional.

Como instruções disponíveis na linguagem LOBAN para a manipulação das construções de dados, podem ser relacionadas as seguintes:

```
INCLUIR <exp-obter-construção> EM <end-ponto>
EXCLUIR DE <end-ponto>
SUBSTITUIR EM <end-ponto> POR <exp-obter-construção>
REPRESENTAR <ex-obter-construção> EM
                                <id-var-PASCAL>
FAZER PARA CADA <end-ponto>
    [EM <exp-obter-construção>]
    ((... <instrução PASCAL/LOBAN> ...))
```

#### c) MENSAGENS

Para o canal de mensagens o sistema utiliza-se de uma variável de estado que informa ao usuário o resultado da execução das operações sobre a base de dados. Esta variável informa o sucesso da operação ou as eventuais ocorrên-

cias de erros que, por ventura, sejam registrados. O conteúdo desta variável poderá ser testado, no programa de aplicação, para que possam ser realizados os procedimentos adequados após a execução de cada operação de acesso ao banco de dados.

Sobre este teste explícito do conteúdo da variável poderá ser montada a estrutura de controle de fluxo.

#### 7.2.3.2 Conversões

Nenhum procedimento de conversão está disponível no SISTEMA L visto que os tipos de PASCAL são compatíveis aos de LOBAN e nas operações de entrada e saída, onde existe a manipulação de variáveis PASCAL nas instruções LOBAN, a comunicação dos dados é realizada a nível de itens elementares como arrays de caracteres, inteiros e reais com as mesmas especificações.

#### 7.2.3.3 Estruturas de controle de fluxo

As estruturas de controle de fluxo para recuperação de tuplas individualmente a partir de um conjunto selecionado são mantidas por uma instrução da DML (LOBAN). Trata-se da instrução "FAZER PARA CADA", que tem a função de produzir uma iteração, onde a cada repetição é recuperada uma construção (ligação, item, tupla) para que sobre esta seja realizado o procedimento adequado. As instruções da linguagem hospedeira que ditam o processamento da construção selecionada serão acopladas à instrução LOBAN "FAZER PARA CADA" conforme pode ser visto pela sintaxe:

```
FAZER PARA CADA <end-ponto>
    [EM <exp-obter-construção>]
    (( ... <instrução LOBAN/PASCAL> ... ))
```



#### 7.2.3.4 Considerações sobre proteção dos dados

A proteção dos dados quanto à atualização concorrente, é oferecida pelo sistema, atualmente, somente ao nível de arquivo e o chaveamento é implícito ao sistema [/HEU 81/].

A proteção dos dados quanto a atualização não autorizada aparece sob a forma de um mecanismo de regulação de acesso obtido mediante as coleções de verbetes de usuário (construções que armazenam a identificação do usuário, senha do usuário e senhas de acesso) e de verbetes de acesso (construções que servem para ligar uma senha de acesso a um, ou dois, termos de acesso que por sua vez faz a ligação entre uma função (GERENCIAR, PROCESSAR, ENDEREÇAR) e os pontos (<end-ponto>) sobre os quais é permitida a função sendo que no protótipo, estes pontos devem ser de arquivos) [/HEU 79/]. Cada vez que um usuário se apresentar ao sistema para utilização de um acervo setorial, o sistema verificará, inicialmente, se existe um verbete de usuário para o usuário em questão, e se os seus dados (senha de usuário e senha de acesso) coincidem com os dados do verbete, além de verificar se existe um verbete de acesso com a senha de acesso fornecida. Caso uma dessas verificações indicar resultado negativo, o usuário terá o seu acesso impedido.

A apresentação do usuário ao sistema é realizada pelo comando USAR ACSET que possui a seguinte sintaxe:

```
USAR ACSET <n ACSET> PARA USUARIO
      [( <senha usuário> )] COM
      <senha acesso> , , ,
```

Quanto a atualização inválida, o sistema oferece a proteção dos dados no sentido de verificar a existência da chave primária (cláusula CONEX) e também controlar o limite de valor de campos nos tipos elementares (limites especificados durante a definição).

#### 7.2.3.5 Considerações sobre homogeneidade

Os padrões de definição da base de dados (DDL) no SISTEMA L são semelhantes aos padrões de definição dos dados em PASCAL, visto que ambas são baseadas em declarações de tipos.

No entanto, com relação a DML embutida na linguagem hospedeira pode-se afirmar que os padrões desta linguagem de manipulação "LOBAN" tem padrões sintáticos próprios e adversos aos padrões dos comandos da linguagem hospedeira PASCAL. Não pode ser, portanto, atribuída uma homogeneidade da DML com a linguagem hospedeira.

#### 7.2.4 Aspectos de implementação do embutimento

##### 7.2.4.1 Descrição da forma de embutimento

A implementação do embutimento da DML LOBAN em programas de aplicação escritos na linguagem PASCAL é realizada através de precompilação.

As instruções LOBAN num programa PASCAL são reconhecidas pelo sinal cardinal (" $\dagger$ ") precedendo estas instruções.

Um programa fonte com instruções PASCAL acrescido de instruções LOBAN é submetido ao precompilador que traduz todas instruções LOBAN em instruções PASCAL correspondentes para que posteriormente o fonte resultante possa ser submetido ao compilador PASCAL.

Para que o processo de tradução, LOBAN para PASCAL, possa ser efetuado, o precompilador utiliza rotinas pré-prontas escritas em PASCAL que estão disponíveis para cada tipo de operação LOBAN realizando as adaptações necessárias a fim de que o código completo de cada uma destas rotinas

nas seja montado a partir das expressões de composição de cada instrução.

O processo completo de tradução pode ser visto na figura 7.3.

#### 7.2.4.2 Considerações sobre a portabilidade da linguagem

Considerando a atual arquitetura do sistema, a portabilidade da linguagem LOBAN com relação a seu embutimento em outras linguagens hospedeiras vai depender do completo projeto e implementação de um novo precompilador específico para a nova linguagem hospedeira.

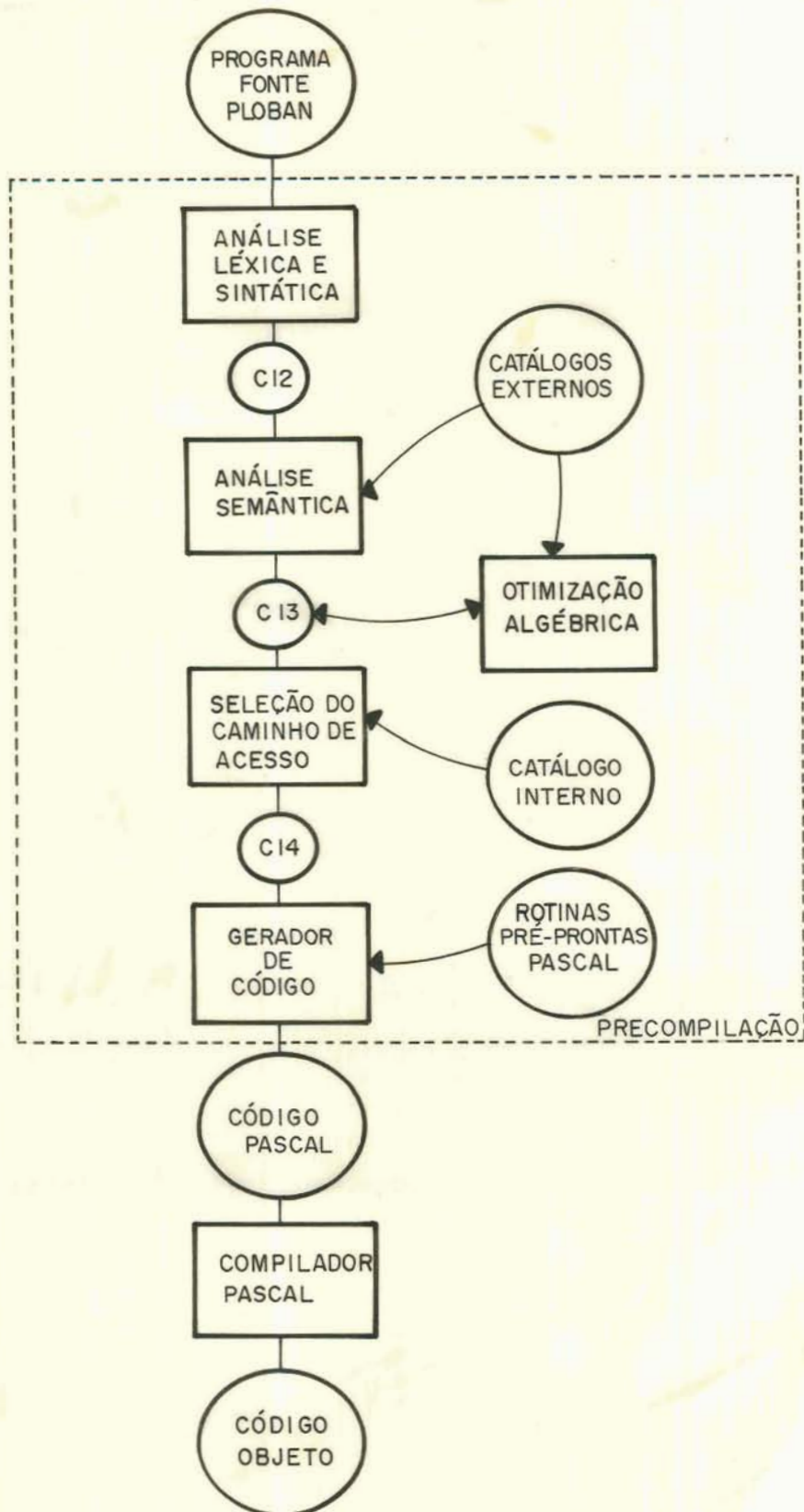


Figura 7.3 - Processo de tradução de LOBAN para PASCAL

### 7.3 Avaliação das características de LOBAN no contexto da classificação

Completada a descrição do SISTEMA L e, por conseguinte, definidas as principais características da linguagem LOBAN, é possível, agora, realizar a avaliação destas características no contexto da classificação projetada no capítulo anterior.

Analisando o quadro sintético de comparação e confrontando as características da linguagem LOBAN com a taxonomia apresentada neste quadro, pode-se delinear o enquadramento desta linguagem neste contexto. Como justificativa, será fornecido, para cada item avaliado, o motivo pelo qual a linguagem se enquadra em determinada classificação.

a) Quanto a definição da área de comunicação de dados:

\* Como o usuário descreve os dados desejados

→. Estruturas e campos são naturalmente manipulados sem necessidade de definição.

MOTIVO: A exemplo do SQL e EQUER, as construções da base de dados definidas anteriormente podem ser referenciadas diretamente no programa sem necessidade de declará-las.

\* Como o usuário recebe os dados desejados

→. Entrada e saída sobre variáveis do programa.

MOTIVO: A instrução REPRESENTAR transfere os dados da base de dados para as variáveis PASCAL e o operador INTERPRETAR realiza o processo inverso.

b) Quanto a criação dinâmica de estruturas do B.D. no programa:

→. Detém a facilidade de geração dinâmica de estruturas.

MOTIVO: A linguagem LOBAN permite a especificação de construções temporárias, mantidas no canal auxiliar.

c) Quanto a conversões de tipos de dados:

→. Nenhum procedimento de conversão é realizado por motivos diversos.

MOTIVO: Tipos de PASCAL são compatíveis aos de LOBAN e a comunicação é realizada a nível de itens elementares.

d) Quanto a homogeneidade:

\* Homogeneidade da DML com a linguagem hospedeira alvo

→. Padrões próprios da DML independentes da linguagem hospedeira.

MOTIVO: As instruções de LOBAN seguem padrões próprios ao S.G.B.D. utilizando, inclusive, uma terminologia voltada à língua portuguesa. A sintaxe difere dos padrões de PASCAL.

\* Homogeneidade entre os embutimentos da DML nas várias linguagens hospedeiras

→. Uniformidade das construções da DML para diversas linguagens hospedeiras.

MOTIVO: Embora esteja, atualmente, disponível somente numa interface com PASCAL, a linguagem

LOBAN assume as características similares ao SQL e EQUER que se propõem a usar padrões únicos independentes da linguagem hospedeira.

e) Quanto ao poder de seleção:

\* Protocolo de recuperação de registros

→. Recuperação de registro único por instrução DML e, também, recuperação de um conjunto de registros por instrução DML.

MOTIVO: A linguagem LOBAN oferece operadores relacionais que selecionam um conjunto de tuplas. Admite também o uso de operações que obtêm uma construção específica (OPERADOR C).

\* Nível das consultas - abrangência de uma consulta

→. Consulta sobre múltiplos conjuntos de dados.

MOTIVO: A linguagem LOBAN possui, em seu acervo de instruções, operadores JUNTAR e LIGAR que obtêm tabelas a partir de duas outras.

\* Nível das consultas - granularidade na informação recuperada

→. Recuperação a nível de qualquer campo no registro.

MOTIVO: A consulta e transferência de dados para o programa de aplicação é realizada informando-se os atributos (campos) de interesse. Estes atributos são transferidos para as variáveis PASCAL através de uma instrução REPRESENTAR.

- \* Nível das consultas - potencialidade existente nas expressões de seleção

→. Seleção com base no valor de qualquer campo.

MOTIVO: A linguagem LOBAN segue uma abordagem de estruturação de dados propícia para a seleção com base na indicação do valor de qualquer campo. Pode ser especificado, nas expressões de obtenção de construção, a seleção pelo conteúdo de qualquer atributo dentro de uma tupla.

- f) Quanto às estruturas de controle de fluxo:

→. Controle de iteração a nível da DML.

MOTIVO: A linguagem LOBAN possui a instrução FAZER PARA CADA que realiza o controle do fluxo para liberação de tuplas individualmente para o programa de aplicação a partir do conjunto de tuplas selecionadas na expressão de obtenção de construção.

- g) Quanto à proteção dos dados:

- \* Atualização concorrente

→. Chaveamento implícito ao sistema.

MOTIVO: Existe, no SISTEMA L, um chaveamento implícito ao nível de arquivo sem a necessidade do usuário especificar instruções de bloqueio para acesso exclusivo à base de dados.

- \* Atualização inválida

→. Controle realizado pelo S.G.B.D.

MOTIVO: O sistema verifica a existência da cha-



ve primária e controla, também, o limite de valores de campos nos tipos elementares.

\* Atualização não autorizada

→. Autorização explícita atribuída ao usuário.

MOTIVO: A regulamentação de acesso é feita mediante o uso de verbetes de usuário para identificar cada usuário que pretende acessar determinada construção e de verbetes de acesso para limitar o tipo de função que o usuário pode executar sobre determinada construção.

h) Quanto ao tratamento de exceções:

→. Através de testes explícitos sobre o conteúdo de mensagens.

MOTIVO: A linguagem LOBAN possui uma variável de estado que indicará as ocorrências durante as operações realizadas sobre o banco de dados. O conteúdo desta variável deverá ser testado a cada instrução para determinar o tipo de procedimento a ser realizado para contornar as eventuais exceções ocorridas.

Percebe-se que a linguagem LOBAN se enquadra no contexto da taxonomia constatada nas demais linguagens da abordagem relacional que são embutidas em linguagens de programação convencionais.

Como visto nas conclusões apresentadas na seção 6.6, pelas próprias características dos S.G.B.D. relacionais, o embutimento é implementado através do processo de precompilação onde as estruturas e campos da base de dados são naturalmente manipulados nos programas de aplicação sem

a necessidade de declará-las neste ambiente e onde a entrada e saída é realizada sobre variáveis do programa.

Outras características dos sistemas de abordagem relacional como facilidade de geração dinâmica de estruturas, protocolo de recuperação de registros baseado na manipulação de conjunto de registros, acesso a múltiplos conjuntos de dados por instrução DML, recuperação a nível de qualquer campo no registro, seleção com base no valor de qualquer campo, além de outras facilidades das operações de álgebra relacional, figuram também na linguagem LOBAN.

Com respeito a conversões de tipos de dados, no entanto, a linguagem LOBAN não está, na presente versão, seguindo os padrões de outras linguagens relacionais como SQL e EQUERL que realizam uma conversão automática a cargo do S.G.B.D. Isto se deve ao fato de que na presente implementação do embutimento existe somente um acoplamento com a linguagem hospedeira PASCAL que utiliza tipos de dados compatíveis com os de LOBAN não necessitando, com isto, procedimentos de conversão. Se, no entanto, embutimentos forem expandidos para outras linguagens hospedeiras, certamente o projeto deverá considerar também o problema de conversões de tipos de dados.

O que poderia ser anexado às facilidades da linguagem LOBAN, certamente sem dificuldades maiores, seria um tratamento de exceções mais aprimorado utilizando, por exemplo, cláusulas específicas para sinalizar os eventos de exceção, como aparece no SQL. Isto evitaria a necessidade do usuário codificar testes explícitos sobre o conteúdo das mensagens a cada operação de acesso ao banco de dados.

Para complementar esta avaliação, aparece ilustrado na figura 7.4 o enquadramento da linguagem LOBAN no contexto do quadro sintético de comparações.

			TOTAL	ADABAS	IMS	IDMS	DMS-II	EQUEL	SQL	PASCAL R	LOBAN	
INTERFACE DE LING. HOSPEDEIRA	IMPLEMENTAÇÃO DO EMBUTIMENTO	CHAMADA À SUBROTINA										
		PRECOMPILAÇÃO DE INSTRUÇÕES DML										
		EXTENSÃO DO COMPILADOR DA LINGUAGEM HOSPEDEIRA										
INTERFACE AUTOCONTIDA												
<b>COMPARAÇÃO A NÍVEL DE DADOS</b>												
COMO O USUÁRIO DEFINE A ÁREA DE COMUNICAÇÃO DE DADOS	COMO O USUÁRIO DESCRIBE OS DADOS DESEJADOS	DESCRIÇÃO AUTOMATICAMENTE INTRODUZIDA NO PROGRAMA										
		O USUÁRIO DEFINE O LAYOUT COMPLETO DAS ESTRUTURAS DESEJADAS										
		O USUÁRIO DEFINE OS CAMPOS DE INTERESSE DE SUA APLICAÇÃO										
	COMO O USUÁRIO RECEBE OS DADOS DESEJADOS	ESTRUTURAS E CAMPOS SÃO NATURALMENTE MANIPULADAS SEM NECESSÁRIA DEFINIÇÃO										
		ENTRADA/SAÍDA FEITA SOBRE A MESMA ÁREA DA DESCRIÇÃO DOS DADOS										
		ENTRADA/SAÍDA EM ÁREAS DECLARADAS PARA ARMAZ. CONTEÚDO (CONCATENADO) DE CAMPOS SOLICITADOS										
CRIAÇÃO DINÂMICA DE ESTRUTURAS DO B. D. DO PROGRAMA	DETÉM A FACILIDADE DE GERAÇÃO DINÂMICA DE ESTRUTURAS											
	NÃO DETÉM A FACILIDADE DE GERAÇÃO DINÂMICA DE ESTRUTURAS											
CONVERSÕES DE TIPOS DE DADOS	CONVERSÃO AUTOMÁTICA A CARGO DO S. G. B. D.											
	CONVERSÃO BASEADA EM INDICAÇÃO EXPLÍCITA DE FORMATO											
	NENHUM PROCEDIMENTO DE CONVERSÃO É REALIZADO POR MOTIVOS DIVERSOS											
<b>COMPARAÇÃO A NÍVEL DE INSTRUÇÕES</b>												
HOMOGENEIDADE	HOMOGENEIDADE DA DML COM A LINGUAGEM HOSPEDEIRA ALVO	UNIFORMIDADE SINTÁTICA ENTRE DML E COMANDOS DA LINGUAGEM HOSPEDEIRA										
		UNIFORMIDADE DE TRATAMENTO DE CHAMADA A INTERFACE S. G. B. D. COM CHAMADA A ROTINAS EXT. CONVENCIONAIS										
		PADRÕES PRÓPRIOS DA DML INDEPENDENTES DOS PADRÕES DA LINGUAGEM HOSPEDEIRA										
HOMOGENEIDADE ENTRE OS EMBUTIMENTOS DA DML NAS VÁRIAS LING. HOSP.	UNIFORMIDADE DAS CONSTRUÇÕES DA DML PARA DIVERSAS LINGUAGENS HOSPEDEIRAS											
	DIFERENCIAÇÃO CONFORME PADRÕES DA LINGUAGEM HOSPEDEIRA											
	RECUPERAÇÃO DE REGISTRO ÚNICO POR INSTRUÇÃO DML											
PODER DE SELEÇÃO	QUANTO AO PROTOCOLO DE RECUPERAÇÃO DE REGISTROS	ASSINALAMENTO DE UM CONJUNTO DE REGISTROS POR INSTRUÇÃO DML										
		RECUPERAÇÃO DE UM CONJUNTO DE REGISTROS POR INSTRUÇÃO DML										
		ABRANGÊNCIA DE UMA CONSULTA	CONSULTA SOBRE CONJUNTO DE DADOS ÚNICO									
	CONSULTA SOBRE MÚLTIPLOS CONJUNTOS DE DADOS											
	CONSULTA SOBRE MÚLTIPLOS CONJUNTOS DE DADOS, PORÉM PREVIA. ASSOCIADOS											
	QUANTO AO NÍVEL DAS CONSULTAS	GRANULARIDADE NA INFORMAÇÃO RECUPERADA	RECUPERAÇÃO A NÍVEL DE REGISTRO									
			RECUPERAÇÃO A NÍVEL DE QUALQUER CAMPO NO REGISTRO									
			RECUPERAÇÃO A NÍVEL DE CAMPOS ESPECÍFICOS									
	POTENCIALIDADE EXISTENTES NAS EXPRESSÕES DE SELEÇÃO	SELEÇÃO COM BASE NO VALOR DE QUALQUER CAMPO										
		SELEÇÃO COM BASE NO VALOR DE DETERMINADO CAMPO										
SELEÇÃO COM BASE EM IDENTIFICAÇÃO INTERNA DO REGISTRO												
SELEÇÃO COM BASE EM CAMINHOS DE ACESSO PRÉ-ESTABELECIDOS												
ESTRUTURAS DE CONTROLE DE FLUXO	CONTROLE DE ITERAÇÃO A NÍVEL DA LINGUAGEM HOSPEDEIRA											
	CONTROLE DE ITERAÇÃO A NÍVEL DA DML											
PROTEÇÃO DOS DADOS	QUANTO A ATUALIZAÇÃO CONCORRENTE	CHAVEAMENTO IMPLÍCITO AO SISTEMA										
		CHAVEAMENTO ESPECIFICADO NA DEFINIÇÃO DOS DADOS										
		ESPEC. EXPLÍCITA DE COMANDOS DML DE CHAVEAMENTO										
		NENHUM CONTROLE ESTÁ IMPLEMENTADO										
	QUANTO A ATUALIZAÇÃO INVÁLIDA	CONTROLE REALIZADO PELO S. G. B. D.										
		CONTROLE EXCLUSIVAMENTE PROGRAMADO PELO USUÁRIO										
	QUANTO A ATUALIZAÇÃO NÃO AUTORIZADA	NENHUMA FACILIDADE IMPLEMENTADA P/ CONTR. ACESSO										
		AUTORIZAÇÃO EXPLÍCITA ATRIBUÍDA AO USUÁRIO										
		AUTORIZAÇÃO IMPLÍCITA NAS VISÕES LOCAIS DO PROG.										
<b>COMPARAÇÃO A NÍVEL DE MENSAGENS</b>												
TRATAMENTO DE EXCEÇÕES	ATRAVÉS DE CLÁUSULAS ESPECÍFICAS PARA SINALIZAR EXCEÇÕES											
	ATRAVÉS DE TESTES EXPLÍCITOS SOBRE O CONTEÚDO DE MENSAGENS											
	ATRAVÉS DE MANIPULAÇÃO AUTOMÁTICA POR ROTINAS DO S. G. B. D.											

Figura 7.4 - A linguagem LOBAN no quadro sintético de comparação das características

## 8. CONCLUSÕES E SUGESTÕES

A dissertação focalizou todos os aspectos relevantes no acoplamento de linguagens de manipulação de dados a linguagens convencionais de programação, considerando as diversas formas de implementar o embutimento. Procurou-se analisar sistematicamente as características dos S.G.B.D. mais expressivos que, direta ou indiretamente, estão relacionados com o embutimento. Desta análise originou a comparação e classificação das características da DML.

A partir deste estudo foi possível verificar as preocupações que surgem no decorrer do projeto de embutimento e constatou-se que nestes projetos existe sempre um interesse em evidenciar o relacionamento que o usuário, como elemento humano no sistema de informação, possui com o S.G.B.D., materializado através da linguagem de manipulação dos dados. Sob este aspecto procurou-se, inclusive, realizar um estudo de alternativas para a interface com o banco de dados. Sob este enfoque, foi possível extrair, também, alguns parâmetros de comparação das linguagens de manipulação de banco de dados.

Constatou-se, também, que a satisfação pessoal do usuário, em utilizar uma determinada linguagem de interface com o S.G.B.D. é, de certa forma, dependente da classe de usuários que dela fará uso para atender às suas necessidades. Um usuário casual, geralmente, se adapta melhor a uma linguagem de consulta interativa, via terminal. Já no caso dos programadores, existe um interesse maior em utilizar facilidades que sejam manipuladas a nível do ambiente do programa de aplicação. Conclui-se, portanto, que é para esta última classe de usuários de um S.G.B.D. que um embutimento de DML em linguagem hospedeira atinge os seus propósitos de implementação.

O assunto de aspectos humanos envolvidos no uso de linguagens, no entanto, assume considerável amplitude in

gressando, inclusive, no campo dos experimentos psicológicos. Em vista deste assunto envolver outras áreas de estudo, trabalhos específicos, que utilizem este enfoque, podem ser desenvolvidos analisando, basicamente, a influência motivacional existente no uso dos diferentes S.G.B.D.

Convém ressaltar, também, que, com o acoplamento de uma linguagem de manipulação de banco de dados a uma linguagem de programação, o usuário estará se deparando com dois ambientes distintos. Numa linguagem de programação convencional, por exemplo, o programa assume o papel de base a partir da qual os dados são trabalhados, enquanto que, num ambiente de banco de dados, os dados assumem o interesse básico do sistema e os programas ocupam uma posição de meros instrumentos para manipular estes dados. Este contraste poderia, inclusive, causar conflitos na disciplina de programação exigindo que o usuário revise a sua metodologia de desenvolvimento de programas. Este é também um assunto que pode ser explorado por um trabalho específico focalizando, principalmente, a forma de compatibilizar ferramentas de linguagens hospedeiras com as ferramentas da linguagem de manipulação de banco de dados. O uso adequado da linguagem de manipulação de banco de dados na linguagem hospedeira vai produzir um bom desempenho do programa e, por conseguinte, do sistema e a isto estará vinculado uma disciplina de programação adequada.

O estudo, aqui descrito, procurou colecionar as informações relevantes no projeto de acoplamento da DML a linguagens de programação pretendendo oferecer algumas ferramentas para futuras implementações de embutimentos ou até para o desenvolvimento de novas linguagens de manipulação de banco de dados.

## BIBLIOGRAFIA

- [ALL 76] ALLMAN, E. & STONEBRAKER, M. Embedding a relational data sublanguage in a general purpose programming language. In: CONFERENCE ON DATA ABSTRACTIONS, DEFINITION AND STRUCTURE, Salt Lake City, Mar. 22-24, 1976. Proceedings. p.25-35. (Em SIGPLAN NOTICES. v.11, special issue,1976).
- [AST 76] ASTRAHAN, M.M. et alii. System R: relational approach to database management. ACM. Transactions on Database Systems, 1(2):97-137, June 1976.
- [BAG 79] BAGGA, C.R.K. & RAJARAMAN, V. A partial implementation of a relational data-base management system as an extension to FORTRAN. Information & Management, 2(6):249-60, Dec. 1979.
- [BEV 80] BEVER, M. & LOCKEMANN, P.C. Embedding data base management systems in programs. Karlsruhe, Universität Karlsruhe, 1980. (Rel. Int. 18/80)
- [BUR 77] BURROUGHS. B7000/B6000 DMS II Host reference manual. Detroit, June 1977.
- [BUR 78] \_\_\_\_\_. B7000/B6000 DMS II DASDL reference manual. Detroit, Mar. 1978.
- [CHA 75] CHAMBERLIN, D.D., GRAY, J.N. & TRAIGER, I.L. Views, authorization and locking in a relational data base system. In: NATIONAL COMPUTER CONFERENCE, Anaheim, May 19-22, 1975. Proceedings. Montvale, AFIPS, 1975. p.425-30. (AFIPS Conference Proceedings, 44)
- [CHA 76] CHAMBERLIN, D.D. et alii. SEQUEL2: A unified approach to data definition, manipulation, and control. IBM J. Res. and Develop., 20(6):560-75, Nov. 1976.
- [CHA 80] CHAMBERLIN, D.D. A summary of user experience with the SQL data sublanguage. In: INTERNATIONAL CONFERENCE ON DATA BASES, Aberdeen, July 1980. Proceedings. s.n.t. p.181-203.
- [CHA 81] CHAMBERLIN, D.D. et alii. A history and evaluation of System R. Communications of the ACM, New York, 24(10):632-46. Oct. 1981.
- [COH 78] COHEN, L.J. Data base management systems. Wellesby, Q.E.D. Information Sciences, 1978.

- [DAL 76] DALE, A.G. & LOWENTHAL, E.I. End-user interfaces for data base management systems. In: SHARE WORKING CONFERENCE ON DATA BASE MANAGEMENT SYSTEMS, 2., Montreal, Apr. 26-30, 1976. Proceedings. Amsterdam, North-Holland, 1977. p. 81-99.
- [DAT 82] DATE, C.J. An introduction to database systems. 3.ed. Addison-Wesley, Reading, Mass. 1982.
- [DAV 77] DAVIS, B. The selection of data-base software. Manchester, NCC, 1977.
- [FUR 82] FURTADO, A.L. & SANTOS, C.S. Organização de Banco de Dados. Rio de Janeiro, Campus, 1982.
- [GOL 81] GOLENDZINER, L.G. & HEUSER, C.A. Executor do código L. Porto Alegre, PGCC da UFRGS, Abr. 1981 (Rel. Técnico 15)
- [GOL 83] GOLENDZINER, L.G., HEUSER, C.A. & IOCHPE, C. Embutimento de LOBAN em PASCAL. Porto Alegre, CPGCC-UFRGS, 1983. (Projeto MINIBAN/UFRGS - Relatório técnico, 31)
- [HEL 75] HELD, G.D., STONEBRAKER, M. & WONG, E. INGRES, a relational data base system. In: NATIONAL COMPUTER CONFERENCE, Anaheim, May 19-22, 1975. Proceedings. Montvale, AFIPS, 1975. p. 409-16. (AFIPS Conference Proceedings, 44)
- [HEU 79] HEUSER, C.A. Estrutura e manipulação da folha: versão 1. Porto Alegre, PGCC da UFRGS, Abr. 1979. (Relatório técnico, 7)
- [HEU 81] HEUSER, C.A., GOLENDZINER, L.G. & OLIVEIRA, J.P.M. Sistema L: uma implementação da linguagem LOBAN. In: SEMINÁRIO INTEGRADO DE SOFTWARE E HARDWARE, 8., Florianópolis, Jul.27-31, 1981. Anais. Florianópolis, UFSC, 1981. p.169-86.
- [HEU 82] HEUSER, C.A. Banco de dados: análise, estrutura de dados e projetos. São Paulo, SCI, Abr.1982.
- [HEU 83] HEUSER, C.A. & GOLENDZINER, L.G. DABOL: a new data base language and its portable implementation. In: WORKSHOP ON RELATIONAL DBMS DESIGN, IMPLEMENTATION, USE ON MICROCOMPUTERS, Toulouse, Feb.14-15, 1983. Proceedings. Rocquencourt, INRIA, 1983.
- [IBM 82a] IBM. SQL/Data System Concepts and Facilities: program product. 2.ed. Endecott, Feb. 1982. (GH24-5013-1)

- [IBM 82b] IBM. SQL/data system for VSE: a relational data system for application development. White Plains, April 1982. (G320-6590-0)
- [JOY 83] JOYCE, J.D. & WARN, D.R. Human factors aspects of a modern data base system. Information & Management, 6(1):27-36, Feb. 1983.
- [LEI 80] LEITE, L.L.P. Introdução aos sistemas de gerência de banco de dados. São Paulo, Edgard Blücher, 1980.
- [MCD 82] McDONALD, N.H. & McNALLY, J.P. Query language feature analysis by usability. Computer Languages, 7(3/4):103-24, 1982.
- [MIC 76] MICHAELS, A.S., MITTMAN, B. & CARLSON, C.R. A comparison of relational and CODASYL approaches to data-base management. Computing Surveys, 8(1):125-51, Mar. 1976.
- [REI 77] REISNER, P. Use of psychological experimentation as an aid to development of a query language. IEEE Transactions on Software Engineering, SE-3(3):218-29, May 1977.
- [REI 81] \_\_\_\_\_. Human factors studies of database query languages: a survey and assessment. Computing Surveys, 13(1):13-32, Mar. 1981.
- [SAN 80] SANTOS, A.C. et alii. Projeto MINIBAN/COPPE especificação da interface LOBAN: linguagem de operação do banco de dados. Rio de Janeiro, COPPE/UFRJ, Out. 1980. v.1 e 2.
- [SCH 77] SCHMIDT, J.W. Some high level language constructs for data of type relation. ACM Transactions on Data Base Systems, 2(3):247-61, Sept. 1977.
- [SCH 80] SCHMIDT, J.M. & MALL, M. Pascal/R report. Hamburg, Universitaet Hamburg, Jan. 1980. (Fachbereich Informatik)
- [SHN 78] SHNEIDERMAN, B. Improving the human factors aspect of database interactions. ACM Transactions on Database Systems, 3(4):417-39, Dec. 1978.
- [SHN 83] \_\_\_\_\_. Human factors of interactive software. In: SCIENTIFIC SYMPOSIUM ON ENDUSER SYSTEMS AND THEIR HUMAN FACTORS, Heidelberg, Mar 18, 1983. Proceedings. Berlin, Springer-Verlag, 1983. p.9-29. (Lecture Notes in Computer Science, 150)



- [SOF 82] SOFTWARE AG. Natural reference manual: version 1.2. s.l., May 1982. (NAT-120-030)
- [STA 74a] STACEY, G.M. A FORTRAN interface to the CODASYL data base task group specifications. Computer Journal, 17(2):124-29, May 1974.
- [STA 74b] \_\_\_\_\_. The interface between a data base and its host language. In: IFIP WORKING CONFERENCE ON DATA BASE MANAGEMENT, Cargèse, Apr.1-5,1974. Proceedings. Amsterdam, North-Holland,1974. p. 305-12.
- [STO 75] STONEBRAKER, M. Implementation of integrity constraints and views by query modification. In: ACM-SIGMOD INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA, San Jose, May 14-16, 1975. Proceedings. San Jose, ACM, 1975. p.65-78.
- [STO 76] STONEBRAKER, M. et alii. The design and implementation of INGRES. ACM Transactions on Database Systems, New York, 1(3):189-222, Sept. 1976.
- [STO 77] STONEBRAKER, M. & ROWE, L.A. Observations on data manipulation languages and their embedding in general purpose programming languages. In: CONFERENCE ON VARY LARGE DATA BASE, 3., Tokyo-Japan, Oct. 6-8, 1977. Proceedings. New York, IEEE, 1977. p.128-43.
- [STO 80] STONEBRAKER, M. Retrospection on a database system. ACM Transactions on Database Systems. New York, 5(2):225-40. June 1980.
- [TAY 76] TAYLOR, R.W. & FRANK, R.L. CODASYL data-base management systems. Computing Surveys, 8(1): 67-104, Mar. 1976.
- [TOT 82] TOTAL Cincom Systems Inc. In: AUERBACH Data Base Management Systems. Pennsauken, 1982. (Rep. 26.01.01)
- [TSI 77] TSICHRITZIS, D.C. & LOCHOVSKY, F.H. Data base management systems. New York, Academic Press, 1977.
- [TSI 78] TSICHRITZIS, D.C. & KLUG, A. The ANSI/X3/SPARC DBMS framework: report of the study group of database management systems. Information Systems, 3(3):173-91, 1978.
- [WEL 81] WELTY, C. & STEMPLE, D.W. Human factors comparison of a procedural and nonprocedural query language. ACM Transactions on Database Systems, 6(4):626-49, Dec. 1981.

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL  
Pós-Graduação em Ciência da Computação

Análise comparativa das formas e  
métodos de embutimento de linguag  
ens para manipulação de banco  
de dados

Dissertação apresentada aos Srs.

*Ligffolenci*  
*Fred W. G. G.*  
*Fred W. G. G.*  
*Paulo*

Visto e permitida a impressão.  
Porto Alegre, 17 / 04 / 84

*Dacilobis G. Landis*  
\_\_\_\_\_  
Coordenador do Curso de Pós-Graduação em  
Ciência da Computação