PUC

Série: Monografias em Ciência da Computação, Nº 07/84

DBTGINHO: UMA IMPLEMENTAÇÃO DA PROPOSTA CODASYL

por

Rubens N. Melo
Sidney Dias da Silva

Departamento de Informática

UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO
MARQUÊS DE SÃO VICENTE, 225 – CEP-22453
RIO DE JANEIRO – BRASIL

PUC/RJ - DEPARTAMENTO DE INFORMÁTICA

Série: Monografias em Ciência da Computação, Nº 07/84
Editor: Antônio L. Furtado Novembro, 1984

DBTGINHO: UMA IMPLEMENTAÇÃO DA PROPOSTA CODASYL*

por

Rubens N. Melo
Sidney Dias da Silva

* Trabalho parcialmente financiado pela FINEP.

RESUMO

Este trabalho apresenta as características básicas do DBTGinho, um sistema de gerência de banco de dados (SGBD) desenvolvido no Departamento de Informática - PUC/RJ. O sistema é do tipo "network", sendo baseado na proposta DBTG-CODASYL (1971).

ABSTRACT

This work presents the general characteristics of DBTGinho, a data base management system (DBMS), developed in the Departamento de Informatica - PUC/RJ. The system is of the "network" type, being based in the DBTG-CODASYL's proposal (1971).

SUMÁRIO

Lista de Figuras

1. INTRODUÇÃO

Neste trabalho é apresentado um resumo das características do sistema DBTGinho, desenvolvido no Departamento de Informática da Pontifícia Universidade Católica do Rio de Janeiro - PUC/RJ.

O DTBGinho é um Sistema de Gerência de Banco de Dados (SGBD), baseado na proposta do Data Base Task Group (DBTG) da Conference on Data Systems Language (CODASYL), de 1971. É do tipo "network" restrito, admitindo relacionamentos do tipo 1:n. Seu desenvolvimento foi iniciado em 1976 e concluído em 1978, sendo implementado no equipamento IBM então disponível na PUC/RJ. Posteriormente, com a mudança verificada no equipamento, o DBTGinho foi revisto e refeito, tendo a sua segunda versão sido implementada no equipamento CDC, em AGO/84.

1.1 - Conceitos Básicos

As definições que serão utilizadas para descrever as estruturas de dados e seus dados são:

a - ítem de dados (data item): a menor unidade de dado conhecida. Uma ocorrência de um ítem de dado é a representação de um valor;

b - dado aritmético (arithmetic data): um ítem de dado aritmético é aquele que tem um valor numérico com características de base, escala e precisão. Os tipos aritméticos incluem ponto fixo (inteiro),

ponto flutuante (real) e complexo;

c - dado em cadeia (string): dados em cadeia podem ser classificados como cadeias de caracteres (character string). O tamanho de um item de dado em cadeia é equivalente ao número de caracteres no item;

d - registro (record): um registro é uma coleção conhecida de zero, um ou mais itens de dados agregados. Pode haver no banco de dados um número arbitrário de ocorrências de registros de cada um dos tipos de registros que foram especificados no "schema" para aquele banco;

e - conjunto (set): um conjunto é uma relação conhecida entre tipos de registro, possuindo características que serão mantidas por todas as suas ocorrências. Cada tipo de conjunto especificado no "schema" tem um tipo de registro declarado como "owner" e um ou mais tipos de registro declarados como "members". Uma ocorrência de conjunto é uma relação entre uma ocorrência de registro "owner" e uma colecao de ocorrências de registros do tipo "member";

f - "schema": a descricao completa do banco de dados; inclui os nomes e descricoes de todas as ocorrências de registro e itens de dados associados que existem no banco, e as descrições dos tipos de ligação entre os tipos de registro.

1.2 - Conjuntos

O SET é o conceito fundamental na estrutura de dados proposta no relatório da CODASYL-DBTG. Como definido anteriormente, o SET é uma relação conhecida entre tipos de registro. O usuário pode especificar no "schema" a ordem do SET, e o "schema" é usado para determinar a ordem na qual as ocorrências de registro MEMBER do SET estarão logicamente armazenadas quando da sua composição.

No DBTGinho os métodos de ordenação são:

a - SORTED: as ocorrências de registro são ordenadas baseadas nos valores da chave especificada. A chave é um item de dados do registro MEMBER;

b - FIRST: as ocorrências de registro são armazenadas de tal forma que a primeira ocorrência de registro criada é a primeira ocorrência de registro acessada (fila);

c - LAST: as ocorrências de registro são armazenadas no SET de tal modo que a última ocorrência de registro criada é a primeira ocorrência de registro acessada (pilha);

d - NEXT: a ocorrência de registro é adicionada ao SET imediatamente após a ocorrência do registro que foi escolhido como referência no momento;

e - PRIOR: a ocorrência de registro é adicionada ao SET imediatamente antes da ocorrência que foi escolhida como referência no momento.

No DBTGinho os SETs são manuais e opcionais: sendo manual, significa que o usuário deve inserir a ocorrência de registro no SET através do uso de comandos DML apropriados; sendo opcional, significa que o MEMBER pode ser modificado, usando os comandos DML.

No DBTGinho os RECORDs têm LOCATION MODE (modo de colocacao no banco de dados) CALC (quando pelo menos um ITEM é assinalado como KEY) ou SYSTEM (quando nenhum ITEM é KEY). Os RECORDs com LOCATION MODE CALC podem ter chaves duplicadas (DUPLICATES ARE ALLOWED).

2. ARQUITETURA DO DBTGINHO

Os componentes básicos do DBTGinho são apresentados na figura 2.1.

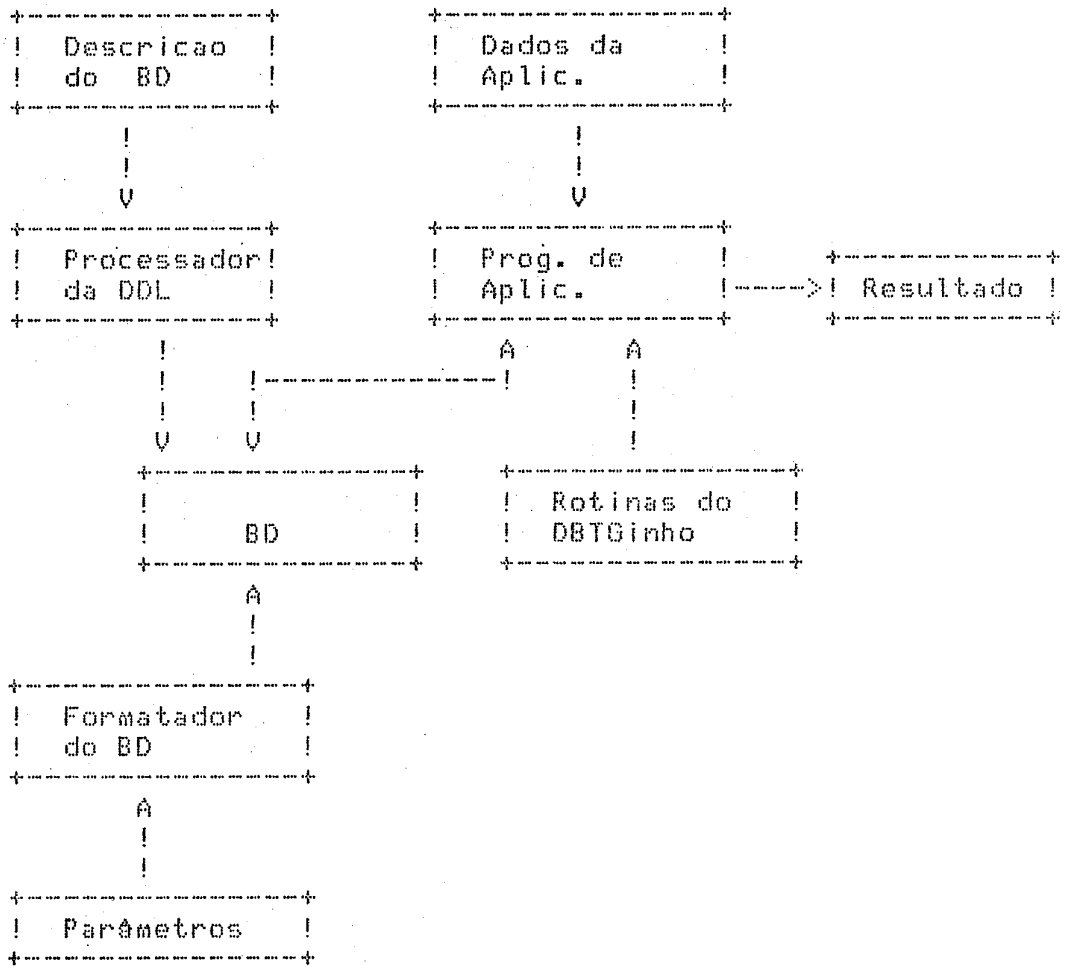

Fig. 2.1 - Arquitetura do DBTGinho

O programa formatador do BD é uma rotina utilitária, que cria, formata e inicializa o arquivo utilizado pelo SGBD.

O processador da DDL é usado para processar a DDL definida pelo usuário e armazená-la sob a forma de um meta-banco de dados (o "schema"). O usuário escreve, então, um programa de aplicação em uma linguagem hospedeira - COBOL ou FORTRAN, a qual deve incluir chamadas ("calls") da DML para o SGBD. O sistema usa o "schema" definido previamente e o BD, determinando como os dados do programa do usuário devem ser interpretados ou retornados.

3. LINGUAGEM DE DESCRICÃO DE DADOS

Uma Linguagem de Descrição de Dados (DDL) é um esquema usado para definir estruturas de dados lógicas. O formato especificado pra o DBTGinho é mostrado na fig. 3.1.

Cartão RECORD

```
Cols.: 01-06 RECORD
       08-11 nome do tipo de registro
```

Cartão ITEM

```
Cols.: 01-04 ITEM
       08-13 nome do item de dado
       15-20 tipo do item de dado
             (INTEGER, CHAR)
       22-27 tamanho do item de dado (bytes)
             tamanhos máximos válidos:
             INTEGER: 10, CHAR: 256
       29-31 indicacão do uso do campo como
             chave (uso da palavra KEY)
```

Cartão SET

```
Cols.: 01-03 SET
       08-13 nome do tipo de set
       15-20 ordem
             (FIRST,LAST,NEXT,PRIOR,SORTED)
       22-27 chave
             (se a ordem é SORTED)
```

Cartão OWNER

```
Cols.: 01-05 OWNER
       08-13 nome do tipo de registro OWNER
```

Cartão MEMBER

```
Cols.: 01-05 MEMBER
       08-13 nome do tipo de registro MEMBER
```

Fig. 3.1 - Formatos da DDL

A DDL consiste de 5 comandos básicos, os quais entram para o analisador DDL como cartões de dados de formato fixo:

a - RECORD: este comando é usado para designar o começo de um novo tipo de registro e definir o seu nome. Todos os cartões ITEM que são colocados após um cartão RECORD pertencem àquele registro;

b - ITEM: este cartão é usado para definir o nome de um item de dado e seus atributos. Estes atributos incluem tipo, tamanho e a indicação ou não do seu uso como chave para a localização do registro;

c - SET: este comando designa o começo de um novo SET, e dá o seu nome, sua ordem e o item de dado que seria a chave de classificacao (sort key), se SORTED. Todos os cartões MEMBER e OWNER que estão associados a um SET devem ser colocados imediatamente após o respectivo cartão SET;

d - OWNER: este cartão define o tipo de registro que é OWNER no SET mencionado no comando SET anterior;

e - MEMBER: este cartão é utilizado pra definir o tipo de registro que será MEMBER do SET mencionado no comando SET anterior.

Algumas restrições quanto à ordem das instruções

DDL são mostradas na fig. 3.2.

1. Os itens que formam um tipo de registro devem seguir imediatamente o cartão RECORD.

2. Os cartões OWNER e MEMBER devem vir imediatamente após o cartão SET.

3. A ordem em que os itens são armazenados no registro é determinada pela ordem na qual os cartões ITEM aparecem após o cartão RECORD.

4. O tipo de registro deve ser definido previamente ao seu uso como OWNER ou MEMBER.

5. O tipo de registro SYSTEM é descrito como um tipo normal.

Fig. 3.2 - Restricões na Ordenacão dos Cartões DDL

Como ilustracão, considere-se a definicão o "schema" de um banco de dados voltado para o armazenamento de dados dos suprimentos em estoque, supridores respectivos e características das pecas estocadas.

A fig. 3.3 apresenta a estrutura de dados lógica para o banco de dados.

Cada ocorrência de registro do tipo SUPD contêm os dados relativos a um supridor (número, nome e cidade); as ocorrências de registro do tipo PART contêm os dados relativos a uma dada peca (número, nome, cor e peso), e as ocorrências de registro do tipo SUPM contêm, cada uma, os dados relativos a um suprimento (número do fornecedor, número da peca e quantidade em estoque).

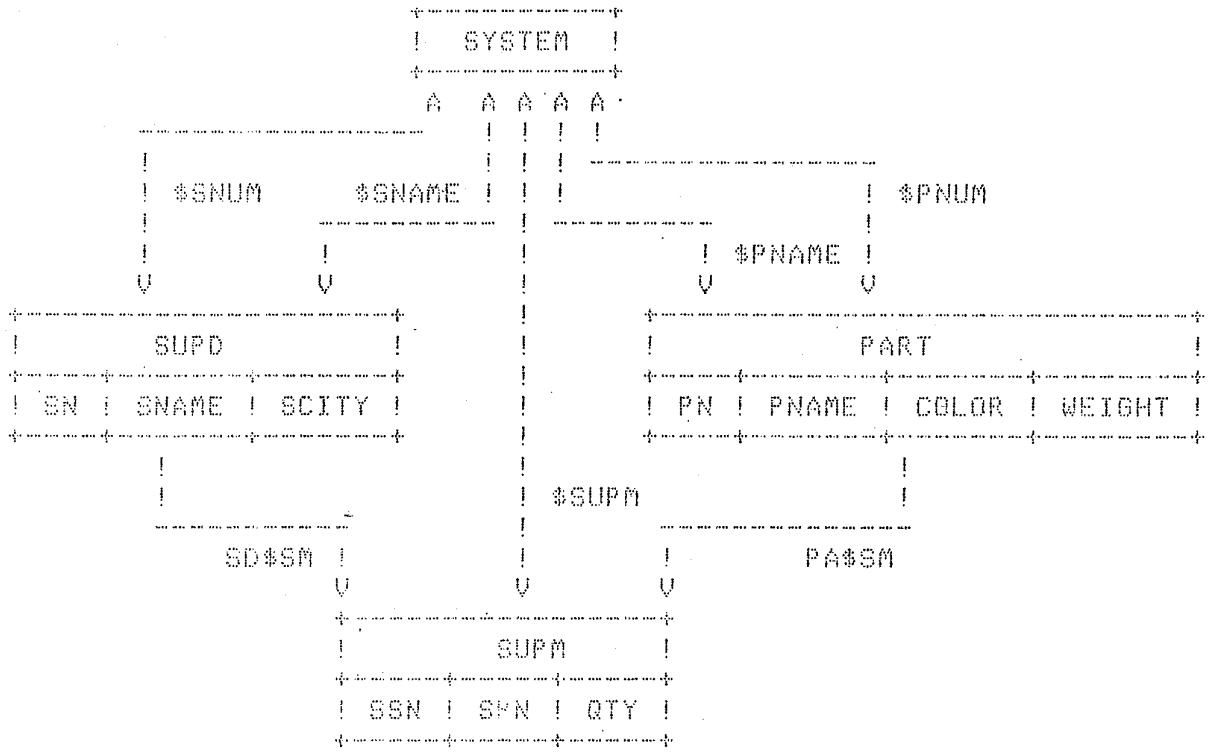

Fig. 3.3 - Estrutura Lógica do Ex. "Suprimentos"

Os SETs formados são:

- $SNUM, cujo OWNER é o SYSTEM, e cujos MEMBERs são ocorrências de registro do tipo SUPD; dessa forma, todas as ocorrências são parte de uma única ocorrência do tipo de SET, estando ordenadas segundo o conteúdo do campo SN;
- $SNAME, com características semelhantes ao SET $SNUM, estando as ocorrências de registro MEMBER ordenadas segundo o conteúdo do campo SNAME;
- $PNUM, cujo OWNER é o SYSTEM, e cujos MEMBERs são ocorrências de registro do tipo PART; dessa

forma, todas as ocorrências são parte de uma única ocorrência do tipo de SET, estando ordenadas segundo o conteúdo do campo PNUM;

- $PNAME, com carcterísticas semelhantes ao SET $PNUM, estado as ocorrências de registro MEMBER ordenadas segundo o conteúdo do campo PNAME;
- $SUPM, cujo owner é o SYSTEM e os MEMBERs são ocorrências de registros do tipo SUPM; dessa forma, todas as ocorrências são parte de uma única ocorrência do tipo de SET;
- SD$SM: as ocorrências deste tipo de SET têm como OWNER uma ocorrência de registro do tipo SUP e como MEMBERs zero ou mais ocorrências de registro do tipo SUPM, ordenadas segundo o conteúdo do campo SPN;
- PA$SM: as ocorrências deste tipo de SET têm como OWNER uma ocorrência de registro do tipo PART e como MEMBERs zero ou mais ocorrências de registro do tipo SUPM, ordenadas segundo o conteúdo do campo SSN.

A DDL para gerar esta estrutura lógica é mostrada na fig. 3.4.

```
*...+....1....+....2....+....3....+....4....+....5

RECORD SYSTEM
RECORD SUPD
ITEM   SN     INTEG  5
ITEM   SNAME  CHAR   30
```

```
ITEM    SCITY   CHAR    15
RECORD  PART
ITEM    PN      INTEG   5
ITEM    PNAME   CHAR    30
ITEM    COLOR   CHAR    10
ITEM    WEIGHT  INTEG   5
RECORD  SUPM
ITEM    SSN     INTEG   5
ITEM    SPN     INTEG   5
ITEM    QTY     INTEG   5
SET     $SNUM   SORTED  SN
OWNER   SYSTEM
MEMBER  SUPD
SET     $SNAME  SORTED  SNAME
OWNER   SYSTEM
MEMBER  SUPD
SET     $PNUM   SORTED  PNUM
OWNER   SYSTEM
MEMBER  PART
SET     $PNAME  SORTED  PNAME
OWNER   SYSTEM
MEMBER  PART
SET     $SUPM   FIRST
OWNER   SYSTEM
MEMBER  SUPM
SET     SD$SM   SORTED  SPN
OWNER   SUPD
MEMBER  SUPM
SET     PA$SM   SORTED  SSN
OWNER   PART
MEMBER  SUPM
```

Fig. 3.4 - DDL Para o Ex. "Suprimentos"

4. INTERFACE COM O USUARIO

Para ser capaz de dirigir a ordem exata na qual as operacões serão realizadas no banco de dados, é necessário o conhecimento detalhado da estrutura de dados lógica. O usuário é também responsável pelo desenvolvimento de algoritmos ou técnicas de caminhamento eficientes através da malha da estrutura de dados. Estas técnicas podem variar de uma estrutura de dados para outra.

4.1 - Indicadores "Currency"

A principal tarefa de um usuário DML é o caminhamento através da estrutura do banco de dados, a qual apresenta problemas semelhantes aos das operacoes de pesquisa de campos em malha.

Se cada ocorrência de registro for vista como um nodo, e cada relacao OWNER-MEMBER no SET for vista como uma ligacão entre nodos, o banco de dados pode ser visto com uma relacão n-dimensional de nodos.

Uma das ferramentas críticas requeridas em qualquer solucão para pesquisa em malha é a capacidade de não perder de vista o ponto onde se está trabalhando. No caso do problema do banco de dados, necessita-se "marcar" de alguma maneira quais os registros que estão sendo usados. O processo usado para marcar as ocorrências de registro desejadas é o de salvar apontadores - ou chaves de banco de dados (DB-Keys) - para as ocorrências desses registros.

A solucão CODASYL consiste em atribuir ao sistema a caracteristica de salvar para o usuário da DML os ponteiros apropriados, não perdendo de vista as ocorrências de registro atuais (currents). Tais ponteiros são conhecidos como indicadores "currency", e podem ser classificados em 3 grupos:

a - CURRENT OWNER: indicador que aponta para a ocorrência de registro que é o "current OWNER" de um tipo de SET particular. Tal indicador é mantido para cada tipo de SET do banco de dados;

b - CURRENT MEMBER: Este indicador aponta para a ocorrência de registro que é o "current MEMBER" de um tipo de SET particular. Tal indicador é mantido para cada tipo de SET no banco de dados. Por definicão, um "current MEMBER" de um tipo de SET deve ter logicamente como OWNER o "current OWNER" desse tipo de SET;

c - CURRENT RECORD: este indicador aponta para o registro corrente que é o "current RECORD" de um dado tipo de registro. Um indicador como este é mantido para cada tipo de registro existente no banco de dados.

4.2 - Linguagem de Manipulacão de Dados

Os comandos DBTGinho consistem de uma colecão de rotinas COBOL que são chamadas para:

- realizar as modificações necessárias nos indicadores "currency";
- extrair e/ou armazenar dados.

Essas rotinas são chamadas de programas COBOL ou FORTRAN, através do uso do comando próprio ("call"). Enquanto a DML cuida da manipulação do banco de dados, as instruções da linguagem hospedeira COBOL ou FORTRAN cuidam dos recursos lógicos e aritméticos. Uma lista dos comandos DML atualmente disponível é mostrada na fig. 4.1. Esses comandos são discutidos nas próximas seções.

Comandos Utilitários	
Abrir o banco de dados para processamento	OPEN
Fechar o banco de dados	CLOS
Comandos Para Criar e Atualizar	
Criar um registro	CR
Criar um registro e armazenar dados	CRS
Adicionar um MEMBER a um SET	AMS
Retirar o "current MEMBER" de um SET	RM
Retirar o SET completo	RS
Comandos de Pesquisa	
Encontrar o primeiro MEMBER de um SET	FFM
Encontrar o último MEMBER de um SET	FLM
Encontrar o próximo MEMBER de um SET	FNM
Encontrar o MEMBER anterior de um SET	FPM
Encontrar o primeiro registro de um tipo	FFR
Encontrar o último registro de um tipo	FLR
Encontrar o próximo registro de um tipo	FNR
Encontrar o registro anterior de um tipo	FPR
Encontrar o MEMBER de um SET com base na chave	FMSK
Encontrar o próximo MEMBER de um SET com a chave	FNSK
Encontrar o registro de um tipo com um dado valor de chave	FFRK
Encontrar o próximo registro com um dado valor de chave	FNRK
Comandos de Manipulacao de Dados	
Apagar o "current MEMBER" de um SET	DRM
Apagar o SET completo	DELS
Obter os dados do registro identificado pela chave	GETK
Obter os dados do "current MEMBER"	GETM
Obter os dados do "current OWNER"	GETO
Obter os dados do "current RECORD"	GETR
Obter o campo do registro identificado pela chave	GFK
Obter o campo do "current MEMBER" de um SET	GFM
Obter o campo do "current OWNER" de um SET	GFO
Obter o campo do "current RECORD" de um tipo	GFR
Atualizar o campo no "current MEMBER" de um SET	SFM
Atualizar o campo no "current OWNER" de um SET	SFO
Atualizar o campo no "current RECORD" de um tipo	SFR
Obter a chave do "current MEMBER" de um SET	GKM
Obter a chave do "current OWNER" de um SET	GKO
Obter a chave do "current RECORD" de um tipo	GKR
Tornar "current MEMBER" de um SET o registro identificado pela chave	SMK
Tornar "current MEMBER" de um SET o "current MEMBER" de outro SET	SMM
Tornar "current MEMBER" de um SET o "current OWNER" de outro SET	SMO
Tornar "current MEMBER" de um SET o "current RECORD" de um tipo	SMR
Tornar "current OWNER" de um SET o "current RECORD" de um tipo	SOR

Tornar o "current OWNER" de um SET o registro identificado pela chave	SOK
Tornar o "current OWNER" de um SET o "current MEMBER" de outro SET	SOM
Tornar o "current OWNER" de um SET o "current OWNER" de outro SET	SOO
Tornar o "current RECORD" o registro identificado pela chave	SRK
Tornar o "current RECORD" o "current OWNER" de um SET	SRO
Tornar o "current RECORD" o "current MEMBER" de um SET	SRM

Fig. 4.1 - Comandos da DML

4.3 - Utilizacao dos Comandos da DML

Os comandos da DML, implementados sob a forma de sub-rotinas, são utilizados através de chamadas específicas. A comunicacao entre o programa e as subrotinas é efetuada através de argumentos. Cada argumento é utilizado como "input" ou "output":se o argumento é de "input", o usuário deve prover o valor; caso o argumento seja utilizado como "output", o sistema retornará um valor.

Por exemplo, a seguinte chamada poderia ser usada, a partir de um programa FORTRAN, para criar uma ocorrência de registro do tipo ALUN:

```
CALL CR(ALUN,KEY,RETCOD)
```

Ao concluir o processamento desse comando o sistema retornaria ao usuário valores para as variáveis KEY e RETCOD. Cada um dos comandos da DML retorna um código de erro ou de mensagem, como seu último parâmetro. Uma lista das mensagens de erro é mostrada na fig. 4.2.

Codigo	Significado
0	execução normal
1	erro de entrada/saída
2	chave inválida/registro inválido/set inválido/item inválido
5	registro não é OWNER válido no conjunto
6	registro não é MEMBER válido no conjunto
8	registro OWNER não posicionado
9	registro MEMBER não posicionado
10	registro do tipo não posicionado
18	conjunto não ordenado
20	número de buffers inválido/modo de uso do BD inválido
21	registro não possui campos-chave
22	registro possui campo-chave
-1	indica que o final de um conjunto foi atingido (não é erro)

Fig. 4.2 - Mensagens de Erro

4.4 - Comandos Para Criar e Atualizar

Estes comandos são usados para:

- criar novas ocorrências de registros;
- adicionar ocorrências de registros a ocorrências SET;
- remover ocorrências de registros de ocorrências de SET.

As duas formas da DML para criar novas ocorrências de registro são:

```
CALL CR(tipo-do-reg,DB-Key,RETCOD)
CALL CRS(tipo-do-reg,DADO,DB-Key,RETCOD)
```

Argumentos: input - tipo-do-reg
DADO (somente na rotina CRS)
output - DB-Key
RETCOD

Currency : output - "current" do tipo-do-reg

Ambos os comandos, CRIAR UM REGISTRO (CR) e CRIAR UM REGISTRO E ARMAZENAR DADOS (CRS),alocam espaco no banco de dados para armazenar uma ocorrência de um tipo especifico de registro. A quantidade de espaco alocado é igual ao tamanho do registro dado. O novo registro torna-se o "current" do tipo, como descrito na DDL, e é retornado para o programador, por meio do argumento DB-Key, a chave do banco de dados que o identifica.

O comando CR não inicializa nenhum dos itens do registro criado; o comando CRS, no entanto, faz isto, armazenando o DADO na ocorrência do registro. E assumido que o conteúdo de DADO obedece à mesma ordem previamente definida na DDL, não sendo feita verificacao para comprovar este fato. Quando o comando CR é usado, caberá ao programador utilizar outros comandos da DML para armazenar dados na ocorrência de registro criada.

Uma vez criada uma ocorrência de registro, o próximo passo consiste em colocá-la, de maneira lógica, na ocorrência de SET adequada. Isto é realizado pelo comando ADICIONAR UM MEMBRO AO SET (AMS).

CALL AMS(tipo-do-set,tipo-do-reg,RETCOD)

Argumentos: input - tipo-do-set
- tipo-do-reg
output - RETCOD

```
Currency  : input  - "current" do tipo-do-reg
                   - "current owner" do tipo-do-set
                   - "current member" do tipo-do-
                     -set
            output - "current member" do tipo-do-
                     -set
```

Este comando adiciona ao tipo de SET especificado, uma ocorrência de registro que é o "current RECORD" de um tipo de registro dado. O "current OWNER" deste tipo de SET será o OWNER do novo tipo de registro, que por sua vez tornar-se-á o "current MEMBER" do tipo de SET. A posição lógica do novo membro no conjunto é determinada pelo critério de seleção especificado na DDL.

Existem dois comandos usados para remover ocorrências de registro de ocorrências de SET. Estes comandos são RETIRAR O "CURRENT MEMBER" DE UM SET (RM) e RETIRAR O SET COMPLETO (RS).

```
CALL RM(tipo-do-set,RETCOD)
CALL RS(tipo-do-set,RETCOD)

Argumentos: input  - tipo-do-set
            output - RETCOD

Currency  : input  - "current owner" do tipo-do-set
                   - "current member" do tipo-do-
                     -set (somente para rotina RM)
            output - "current member" do tipo-do-
                     -set
```

Usando o comando RM, é removido do SET de maneira lógica a ocorrência de registro que é o "current MEMBER" do tipo-de-set especificado, isto é, o membro anterior e o membro posterior a este são ligados entre si. O "current MEMBER" passa a

ser a ocorrência lógica do membro seguinte ao removido. No caso de haver sido removido o último membro, o "current MEMBER" é tornado indefinido.

O comando RS retira, de maneira lógica, todas as ocorrências de membros do SET cujo tipo foi dado como argumento. O "current MEMBER" é tornado nulo.

4.5 - Comandos de Pesquisa

Este grupo de comandos é usado para fazer o caminhamento através da estrutura de dados. Os comandos são usados para seguir os apontadores lógicos dentro da ocorrência de um SET. Os primeiros dois comandos tratados, ENCONTRAR O PRIMEIRO MEMBRO DE UM SET (FFM) e ECONTRAR O ÚLTIMO MEMBRO DE UM SET (FLM), são usados para obter a entrada inicial na ocorrência do SET:

```
CALL FFM(tipo-do-set,RETCOD)
CALL FLM(tipo-do-set,RETCOD)

Argumentos: input  - tipo-do-set
            output - RETCOD

Currency  : input  - "current owner" do tipo-do-set
            output - "current member" do tipo-do-
                     -set
```

O comando FFM faz com que o primeiro membro lógico de um dado tipo de SET seja o "current MEMBER" deste SET. O comando FLM possui efeito semelhante, tornando o último membro o "current MEMBER" do SET. Ambos os comandos necessitam que exista um "current OWNER". Caso nao existam membros no SET, é retornado

o código indicativo de final de conjunto (-1).

Uma vez que o usuário tenha entrado inicialmente na ocorrência de SET com FFM ou FLM, ele poderá querer mover-se para frente ou para trás (isto de maneira lógica), percorrendo a estrutura do SET. Isto é realizado pelas rotinas ENCONTRAR O PROXIMO MEMBRO EM UM SET (FNM) e ENCONTRAR O MEMBRO ANTERIOR EM UM SET (FPM):

```
CALL FNM(tipo-do-set,RETCOD)
CALL FPM(tipo-do-set,RETCOD)

Argumentos: input  - tipo-do-set
            output - RETCOD

Currency  : input  - "current owner" do tipo-do-set
            output - "current member" do tipo-do-
                     -set
```

O comando FNM faz com que o próximo membro logicamente posterior ao atual "current MEMBER" do tipo de SET dado torne-se o "current MEMBER". O comando FPM possui efeito semelhante, tornando o membro logicamente anterior ao "current MEMBER" do tipo de SET o "current MEMBER". Caso não exista o membro a ser posicionado (anterior ou próximo), é retornado o código indicativo de final de conjunto (-1).

O último tipo de comando de pesquisa em conjuntos inclui algum processamento lógico. A rotina ENCONTRAR O MEMBRO DE UM SET COM BASE NA "SORT KEY" (FMSK), procura através de um SET que está classificado e localiza a ocorrência de registro que contêm o valor especificado pela "sort key".

```
CALL FMSK(tipo-do-set,valor-da-sort-key,RETCOD)

Argumentos: input  - tipo-do-set
                   - valor-da-sort-key
            output - RETCOD

Currency  : input  - "current owner" do tipo-do-set
            output - "current member" do tipo-do-
                     -set
```

A ocorrência de registro que é localizada com valor igual ao da "sort key" é tornada o "current MEMBER" deste SET. Caso tal membro não seja localizado, o "current MEMBER" é tornado indefinido, e o código indicativo de final de conjunto é retornado (-1).

A outra rotina do tipo é a ENCONTRAR O PROXIMO MEMBRO DE UM SET COM A CHAVE (FNSK); que possibilita a localizacao da próximo membro lógico do conjunto que possui o mesmo valor de "sort key".

```
CALL FNSK(tipo-do-set,valor-da-sort-key,RETCOD)

Argumentos: input  - tipo-do-set
                   - valor-da-sort-key
            output - RETCOD

Currency  : input  - "current owner" do tipo-do-set
                   - "current member" do tipo-do-
                     -set
            output - "current member" do tipo-do-
                     -set
```

E localizada a ocorrência de registro com valor igual ao da "sort key", e que logicamente seja posterior à

ocorrência que é o "current MEMBER", tornando-se, por sua vez, o "current MEMBER" deste SET. Caso a ocorência não seja localizada, o código indicativo de final de conjunto é retornado (-1).

Existe, também, um grupo de comandos utilizados para seguir os apontadores lógicos dentro das ocorrências de um tipo de registro. Os comandos deste grupo são semelhantes aos anteriormente examinados, relativos á localização de ocorrências de registros no contexto de conjuntos.

Os comandos ENCONTRAR O PRIMEIRO REGISTRO DE UM TIPO (FFR) e ENCONTRAR O ULTIMO REGISTRO DE UM TIPO (FLM), são usados para obter a entrada inicial nas ocorrências de registro do tipo:

```
CALL FFR(tipo-do-reg,RETCOD)
CALL FLR(tipo-do-reg,RETCOD)

Argumentos: input  - tipo-do-reg
            output - RETCOD

Currency  : input  - nenhum
            output - "current record" do tipo-do-
                     -reg
```

O comando FFR faz com que a primeira ocorrência lógica de um dado tipo de registro seja o "current RECORD" deste tipo. O comando FLR possui efeito semelhante, tornando o último membro o "current RECORD" do tipo. Em ambos os casos, na hipótese de não existir ao menos uma ocorrência de registro, é retornado o código indicativo de final de conjunto (-1).

O caminhamento no contexto das ocorrências de um

dado tipo de registro, para frente ou para trás (de maneira lógica), pode ser feito através dos comandos ENCONTRAR O PROXIMO REGISTRO DE UM TIPO (FNR) e ENCONTRAR O REGISTRO ANTERIOR DE UM TIPO (FPR):

```
CALL FNR(tipo-do-reg,RETCOD)
CALL FPR(tipo-do-reg,RETCOD)

Argumentos: input  - tipo-do-reg
            output - RETCOD

Currency   :input  - "current record" do tipo-do-
                      -reg
            output - "current record" do  tipo-do-
                      -reg
```

O comando FNR faz com que a próxima ocorrência lógicamente posterior ao atual "current RECORD" do tipo de registro torne-se o "current RECORD". O comando FPR possui efeito semelhante, tornando a ocorrência de registro anterior ao "current RECORD" o novo "current RECORD". Na hipótese de não haver ocorrência de registro próxima ou anterior, o código indicativo de final de conjunto é retornado (-1).

Com o uso da rotina ENCONTRAR O REGISTRO DO TIPO COM UM DADO VALOR DE CHAVE, a localizacão de uma ocorrência de registro pode ser feita diretamente, a partir do valor da sua chave. A chave, no caso, é composta pelos campos com atributo KEY, previamente declarados na DDL:

```
CALL FFRK(tipo-do-reg,valor-da-chave,RETCOD)

Argumentos: input  - tipo-do-reg
                   - valor-da-chave
            output - RETCOD
```

```
Currency  : input  - nenhum
            output - "current RECORD" do tipo-do-
                     -reg
```

A ocorrência de registro com valor igual ao da chave é tornada o "current RECORD" do tipo. Caso não seja localizada qualquer ocorrência, o "current RECORD" é tornado indefinido, e o código indicativo de final de conjunto é retornado (-1).

A próxima ocorrência, com o mesmo valor de chave, pode ser posicionada com o uso da rotina ENCONTRAR O PROXIMO REGISTRO COM O MESMO VALOR DE CHAVE (FNRK):

```
CALL FNRK(tipo-do-reg,valor-da-chave,RETCOD)

Argumentos: input  - tipo-do-reg
                   - valor-da-chave
            output - RETCOD

Currency  : input  - "current record" do tipo-do-
                     -reg
            output - "current record" do tipo-do-
                     -reg
```

A ocorrência de registro com valor igual ao da chave, e logicamente posterior ao "current RECORD", torna-se o "current RECORD". Caso não seja localizada qualquer ocorrência com o valor, o "current RECORD" torna-se indefinido, e o código indicativo de final de conjunto é retornado (-1).

4.6 - Comandos de Manipulação de Dados

Neste grupo, diversos comandos da DML têm quatro formatos diferentes,que podem ser usados pelo programdor. A forma

básica realiza a manipulação sobre ocorrências de registros, com base na chave do banco de dados informada. As outras três formas realizam manipulações com base no "current MEMBER" do SET, no "current OWNER" do SET, ou no "current RECORD" do tipo de registro. Como geralmente o usuário não conhece a atual chave do banco de dados, os indicadores "currency" devem ser usados para determinar a chave para uma desejada ocorrência de registro.

Os comandos a seguir permitem ao usuário buscar uma chave de banco de dados para uma ocorrência de registro que é um "current OWNER", "current MEMBER" ou um "current RECORD":

```
CALL GKM(tipo-do-set,DBKEY,RETCOD)
CALL GKO(tipo-do-set,DBKEY,RETCOD)
CALL GKR(tipo-do-reg,DBKEY,RETCOD)

Argumentos: input  - tipo-do-set (GKM e GKO)
                   - tipo-do-reg (GKR)
            output - DBKEY
                     RETCOD

Currency  : input  - "current owner" do tipo-de-set
                     (GKO)
                   - "current -member" do  tipo-de-
                     -set (GKM)
                   - "current  record" do  tipo-de-
                     -reg (GKR)
```

A rotina GKM devolve, por meio da DBKEY, o valor da chave do banco de dados referente ao "current MEMBER" do tipo de SET. As rotinas GKO e GKR têm comportamente semelhantes, retornando, respectivamente, as chaves correspondentes aos "current OWNER" e "current RECORD".

As ocorrências de registro podem ser retiradas fisicamente do banco de dados, através dos comandos da DML APAGAR O "CURRENT MEMBER" DO SET (DRM) e APAGAR O SET COMPLETO (DELS):

```
CALL DRM(tipo-do-set,RETCOD)
CALL DELS(tipo-do-set,RETCOD)

Argumentos: input  - tipo-do-set
            output - RETCOD

Currency  : input  - "current owner" do tipo-de-set
                     (DELS)
                   - "current member" do tipo-de-
                     -set (DRM)
            output - "current member" do tipo-de-
                     -set indefinido
                   - "current owner" do tipo-de-set
                     indefinido (DELS)
```

No caso da rotina DRM, a ocorrência de registro que é o "current MEMBER" do SET é retirada fisicamente do banco de dados. Isto implica em que, previamente, a ocorrência seja desligada de todas as ocorrências de SETs de que participa como MEMBER, e que todas as ocorrências de SETs em que é OWNER sejam desfeitas, isto é, que todos os MEMBERs sejam desligados.

O procedimento é repetido, no caso da rotina DELS, para todos os MEMBERs da ocorrência do SET identificada pelo "current OWNER".

Os comandos OBTENHA OS DADOS DO REGISTRO COM BASE NA CHAVE (GETK). e OBTENHA O CAMPO DE UM REGISTRO COM BASE NA CHAVE (GFK), são usados para buscar dados de uma ocorrência de registro especifica:

```
CALL GETK(DBKEY,DADO,RETCOD)
CALL GFK(tipo-do-item,DBKEY,DADO,RETCOD)

Argumentos: input  - tipo-do-item (GFK)
                   - DBKEY
            output - DADO
                   - RETCOD

Currency  : nenhum
```

O comando GETK é usado pra recuperar todos os Itens de dados de uma ocorrência de registro particular, associada a uma dada DBKEY. Os valores retornam por meio do argumento DADO, tendo a mesma ordem definida na DDL. O comando GFK recupera somente o valor para o item de dado especificado e existente no registro identificado pela DBKEY. Em ambas as situações não é feito teste do tamanho do argumento DADO, para determinar se o mesmo é ou não suficiente para receber os dados desejados.

Tais comandos, juntamente com as variantes a seguir, dão ao usuário a capacidade de recuperar dados:

```
a - CALL GETM(tipo-do-set,DADO,RETCOD)
b - CALL GETO(tipo-do-set,DADO,RETCOD)
c - CALL GETR(tipo-do-reg,DADO,RETCOD)
d - CALL GFM(tipo-do-item,tipo-do-set,DADO,RETCOD)
e - CALL GFO(tipo-do-item,tipo-do-set,DADO,RETCOD)
f - CALL GFR(tipo-do-item,tipo-do-reg,DADO,RETCOD)
```

Cada comando equivale, de maneira lógica, a outros comandos:

```
a - CALL GKM(tipo-do-set,DBKEY,RETCOD)
    CALL GETK(DBKEY,DADO,RETCOD)
```

```
b - CALL GKO(tipo-do-set,DBKEY,RETCOD)
    CALL GETK(DBKEY,DADO,RETCOD)
c - CALL GKR(tipo-do-reg,DBKEY,RETCOD)
    CALL GETK(DBKEY,DADO,RETCOD)
d - CALL GKM(tipo-do-set,DBKEY,RETCOD)
    CALL GFK(tipo-do-item,DBKEY,DADO,RETCOD)
e - CALL GKO(tipo-do-set,DBKEY,RETCOD)
    CALL GFK(tipo-do-item,DBKEY,DADO,RETCOD)
c - CALL GKR(tipo-do-reg,DBKEY,RETCOD)
    CALL GFK(tipo-do-item,DBKEY,DADO,RETCOD)
```

Os comandos ATUALIZAR O CAMPO NO "CURRENT MEMBER" DO SET, ATUALIZAR O CAMPO NO "CURRENT OWNER" DO SET e ATUALIZAR O CAMPO NO "CURRENT RECORD" DO TIPO, possuem efeitos inversos aos das rotinas GFM, GFO e GFR, respectivamente:

```
CALL SFM(tipo-do-item,tipo-do-set,DADO,RETCOD)
CALL SFO(tipo-do-item,tipo-do-set,DADO,RETCOD)
CALL SFR(tipo-do-item,tipo-do-reg,DADO,RETCOD)

Argumentos: input  - tipo-do-item
                   - tipo-do-set (GFM e GFO)
                   - tipo-do-reg (GFR)
                   - DADO
            output - RETCOD

Currency :  input  - "current member" do  tipo-de-
                     -set (SFM)
                   - "current owner" do tipo-de-set
                     (SFO)
                   - "current record" do  tipo-de-
                     -reg (SFR)
```

O valor do DADO é armazenado no item de dado especificado da ocorrência de registro que for o "current" do SET ou registro, permitindo ao usuário mudar o valor de um item de dado, bem como armazenar novos valores em novos registros. Cuidados especiais devem ser tomados quando atualizando o conteúdo de itens de dado declarados como chaves.

Existe um conjunto de comandos que não são voltados

para o acesso e o armazenamento de ocorrências de registros de dados. Tais comandos cuidam do problema do caminhamento na rede, e possibilitam a modificação dos indicadores "currency". Os comandos ATUALIZE O "CURRENT MEMBER" COM BASE NA CHAVE (SMK), ATUALIZE O "CURRENT OWNER" COM BASE NA CHAVE (SOK) e ATUALIZE O "CURRENT RECORD" COM BASE NA CHAVE (SRK) são usados para fazer com que uma ocorrência de registro identificada pela chave seja, respectivamente, tornada o "current MEMBER", "current OWNER" ou o "current RECORD" do tipo de SET ou registro especificado.

```
CALL SMK(tipo-do-set,DBKEY,RETCOD)
CALL SOK(tipo-do-set,DBKEY,RETCOD)
CALL SRK(tipo-do-reg,DBKEY<RETCOD)

Argumentos: input  - tipo-do-set (SMK,SOK)
                   - tipo-do-reg (SRK)
            output - DADO
                   - RETCOD

Currency :  output - "current member" do  tipo-de-
                     -set (SFM)
                   - "current owner" do tipo-de-set
                     (SFO)
                   - "current record" do  tipo-de-
                     -reg (SFR)
```

Existem outras variações à disposição do usuário:

```
CALL SMM(tipo-do-set-1,tipo-do-set-2,RETCOD)
CALL SMO(tipo-do-set-1,tipo-do-set-2,RETCOD)
CALL SOM(tipo-do-set-1,tipo-do-set-2,RETCOD)
CALL SOO(tipo-do-set-1,tipo-do-set-2,RETCOD)

Argumentos: input  - tipo-do-set-1
                   - tipo-do-set-2
            output - RETCOD

Currency :  input  - "current member" do  tipo-de-
                     -set-2 (SMM)  (SOM)
                   - "current owner" do tipo-de-set
```

```
                    -2 (SMO) (SOO) .
          output - "current member" do  tipo-de-
                   -set-1 (SMM)  (SMO)
                 - "current owner" do tipo-de-set
                   -2 (SOM) (SOO)
```

A rotina SMM possibilita tornar "current MEMBER" do tipo-de-set-1 o "current MEMBER" do tipo-de-set-2; A rotina SMO possibilita, por seu turno, que o "current OWNER" do tipo-de-set--2 seja tornado o "current MEMBER" do tipo-de-set-1. As rotinas SOM e SOO têm efeitos semelhantes, no que se refere ao estabelecimento do "current OWNER" do tipo-de-set-1.

O usuário pode, também, alterar os indicadores "currency" dos SETs com base nos "currents" dos tipos de registro, e vice-versa, através das rotinas:

```
CALL SMR(tipo-do-set,tipo-do-reg,RETCOD)
CALL SOR(tipo-do-set,tipo-do-reg,RETCOD)
CALL SRM(tipo-do-reg,tipo-do-set,RETCOD)
CALL SOO(tipo-do-reg,tipo-do-set,RETCOD)

Argumentos: input  - tipo-do-reg
                   - tipo-do-set
            output - RETCOD

Currency :  input  - "current record" do  tipo-de-
                     -reg (SMR)  (SOR)
                   - "current owner" do tipo-de-set
                     (SRO)
                   - "current member" do  tipo-de-
                     -set (SRM)
            output - "current member" do  tipo-de-
                     -set (SMR)
                   - "current owner" do tipo-de-set
                     -(SOR)
                   - "current  record" do  tipo-de-
                     -reg (SRM) (SRO)
```

Através do uso das rotinas SMR e SMO, o "current

RECORD" de um tipo de registro pode ser tornado o "current MEMBER" ou o "current OWNER" de um tipo de SET. As rotinas SRM e SRO possibilitam a obtencão de efeito inverso.

Com esses comandos, o usuário do DBTGinho pode trabalhar com estruturas de vários níveis, utilizando as rotinas da DML para fazer o caminhamento pela rede.

4.7 - Comandos Utilitários

Os comandos utiliários são usados para abrir e fechar o banco de dados.

O comando OPEN é usado para abrir o banco de dados e inicializar as variáveis de uso do sistema:

```
CALL OPEN(NUMPAG,MODO,RETCOD)

Argumentos: input  - NUMPAG
                   - MODO
            output - RETCOD

Currency :  nenhum
```

O argumento NUMPAG deve conter um número inteiro, pertencente ao intervalo 1-10, que indica o número de páginas que deve ser alocado pelo sistema. Quanto maior o número de páginas, maior a quantidade de memória principal ocupada pelo programa e menor o tempo dispendido com leitura e/ou gravacão de dados pelo DBTGinho.

O argumento MODO deve conter "READ" ou "WRITE", indicando como será utilizado o arquivo do banco de dados. Caso

se deseje ler e alterar dados, o modo "WRITE" deve ser especificado na abertura do banco. Quando utilizando o banco de dados com MODO "READ", o valor mínimo para NUMPAG é 3.

O comando CLOS é utilizado para fechar um banco de dados previamente aberto.

CALL CLOS(RETCOD)

Argumentos: output - RETCOD

Currency : nenhum

---

5. BIBLIOGRAFIA

Data Base Task Group of CODASYL Programming Language Commitee,
Report (April 1971),ACM,2nd. printing,NY,1975.

Crisostomo, C. P.,
Manual do DBTGinho,
Relatório de Trabalho Individual,DI-PUC/RJ,1979.

APENDICE A - EXEMPLO DE DEFINICAO DE SCHEMA

CHAMADA DO UTILITARIO E COMANDOS DA DDL

```
SCHEMA.
USER,USUARIO,SENHA.
CHARGE,CONTA,USUARIO.
ATTACH,DBMSLIB/UN=KAPINF5.
ATTACH,BANCO/M=W.
LIBRARY,DBMSLIB.
DBCOMP.
*FOR
RECORD SUPD
ITEM    SNUM    INTEG  5
ITEM    SNAME   CHAR   30
ITEM    SCITY   CHAR   15
RECORD PART
ITEM    PNUM    INTEG  5
ITEM    PNAME   CHAR   30
ITEM    COLOR   CHAR   10
ITEM    WEIGHT  INTEG  5
RECORD SUPM
ITEM    SSN     INTEG  5
ITEM    SPN     INTEG  5
ITEM    QTY     INTEG  5
RECORD SYSTEM
SET     $SNUM   SORTED SNUM
OWNER   SYSTEM
MEMBER  SUPD
SET     $SNAME  SORTED SNAME
OWNER   SYSTEM
MEMBER  SUPD
SET     $PNUM   SORTED PNUM
OWNER   SYSTEM
MEMBER  PART
SET     $PNAME  SORTED PNAME
OWNER   SYSTEM
MEMBER  PART
SET     $SUPM   FIRST
OWNER   SYSTEM
MEMBER  SUPM
SET     SD$SM   SORTED SPN
OWNER   SUPD
MEMBER  SUPM
SET     PA$SM   SORTED SSN
OWNER   PART
MEMBER  SUPM
*FOR
*EOF
```

LISTAGEM DOS COMANDOS DDL E DO SCHEMA GERADO

DBTG - VERSAO 2.1 - DBCOMP PAG. 1

```
SEQ.    COMANDO
-------  ------------------------------------

    1   RECORD  SUPD
    2   ITEM    SNUM    INTEG   5
    3   ITEM    SNAME   CHAR    30
    4   ITEM    SCITY   CHAR    15
    5   RECORD  PART
    6   ITEM    PNUM    INTEG   5
    7   ITEM    PNAME   CHAR    30
    8   ITEM    COLOR   CHAR    10
    9   ITEM    WEIGHT  INTEG   5
   10   RECORD  SUPM
   11   ITEM    SSN     INTEG   5
   12   ITEM    SPN     INTEG   5
   13   ITEM    QTY     INTEG   5
   14   RECORD  SYSTEM
   15   SET     $SNUM   SORTED  SNUM
   16   OWNER   SYSTEM
   17   MEMBER  SUPD
   18   SET     $SNAME  SORTED  SNAME
   19   OWNER   SYSTEM
   20   MEMBER  SUPD
   21   SET     $PNUM   SORTED  PNUM
   22   OWNER   SYSTEM
   23   MEMBER  PART
   24   SET     $PNAME  SORTED  PNAME
   25   OWNER   SYSTEM
   26   MEMBER  PART
   27   SET     $SUPM   FIRST
   28   OWNER   SYSTEM
   29   MEMBER  SUPM
   30   SET     SD$SM   SORTED  SPN
   31   OWNER   SUPD
   32   MEMBER  SUPM
   33   SET     PA$SM   SORTED  SSN
   34   OWNER   PART
   35   MEMBER  SUPM
```

DRTG - VERSAO 2.1 - DBCOMP PAG. 2

****** SCHEMA ******

REGISTROS

NOME	TAM. DADOS	DESL DADOS	N.SETS OWNER	N.SETS MEMBER	ITEM DADO	TAM. ITEM	TIPO ITEM	CHAVE (S/N)
SUPD	50	130	1	2	SNUM	5	INTEG	N
					SNAME	30	CHAR	N
					SCITY	15	CHAR	N
PART	50	130	1	2	PNUM	5	INTEG	N
					PNAME	30	CHAR	N
					COLOR	10	CHAR	N
					WEIGHT	5	INTEG	N
SUPM	15	130		3	SSN	5	INTEG	N
					SPN	5	INTEG	N
					QTY	5	INTEG	N

SETS

NOME	OWNER	MEMBER	TIPO	CHAVE	DSL.PT OWNER	DSL.PT MEMBER
$SNUM	SYSTEM	SUPD	SORTED	SNUM	4	4
$SNAME	SYSTEM	SUPD	SORTED	SNAME	7	7
$PNUM	SYSTEM	PART	SORTED	PNUM	10	4
$PNAME	SYSTEM	PART	SORTED	PNAME	13	7
$SUPM	SYSTEM	SUPM	FIRST		16	4
SD$SM	SUPD	SUPM	SORTED	SPN	10	7
PA$SM	PART	SUPM	SORTED	SSN	10	10

APENDICE B - EXEMPLOS DE PROGRAMAS

EXEMPLO 1: CARGA DO BANCO DE DADOS

```
CARGABD.
USER,USUARIO,SENHA.
CHARGE,CONTA,USUARIO.
ATTACH,DBMSLIB/UN=KAPINF5.
ATTACH,BANCO/M=W.
LIBRARY,DBMSLIB.
COBOL5.
LGO.
*EOR
       IDENTIFICATION DIVISION.
       PROGRAM-ID. CARGABD.
      *-----------------------------------------------------------------*
      *  FUNCAO: CARREGAR O BANCO DE DADOS COM REGISTROS DE SUPRIDO-    *
      *          RES, PECAS E SUPRIMENTOS.                              *
      *-----------------------------------------------------------------*
      *
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SPECIAL-NAMES.
           #1# IS SALTO.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT ENTRADA ASSIGN TO #INPUT#.
           SELECT SAIDA   ASSIGN TO #OUTPUT#.
      *
       DATA DIVISION.
       FILE SECTION.
      *
       FD  ENTRADA LABEL RECORDS ARE OMITTED
                   BLOCK CONTAINS 1 RECORDS.
       01  REG-ENTRADA.
           05 TIPO PIC X.
           05 LAY-OUT-SUPD.
              09 SN           PIC 9(5).
              09 SNAME        PIC X(30).
              09 SCITY        PIC X(15).
              09 FILLER       PIC X(29).
           05 LAY-OUT-PART REDEFINES LAY-OUT-SUPD.
              09 PN           PIC 9(5).
              09 PNAME        PIC X(30).
              09 COLOR        PIC X(10).
              09 WEIGHT       PIC 9(5).
              09 FILLER       PIC X(29).
           05 LAY-OUT-SUPM REDEFINES LAY-OUT-SUPD.
              09 SSN          PIC 9(5).
              09 SPN          PIC 9(5).
              09 QTY          PIC 9(5).
```

```
        09 FILLER       PIC X(64).
*
 FD  SAIDA    LABEL RECORDS ARE OMITTED
              BLOCK CONTAINS 1 RECORDS.
 01  LINHA                PIC X(132).
*
 WORKING-STORAGE SECTION.
 01  CHAVE-FIM                  PIC  9(01) COMP-1 VALUE 0.
     88 EOF                                       VALUE 1.
 01  CONT-LIN                   PIC S9(02) COMP-1 VALUE +60.
 01  NUMPAG                     PIC S9(02) COMP-1 VALUE +10.
 01  RET-CODE                   PIC S9(03) COMP-1 VALUE +0.
 01  MODO                       PIC X(10) VALUE "WRITE".
 01  NOMREG                     PIC X(08).
 01  NOMSET                     PIC X(08).
 01  NOMSET-1                   PIC X(08).
 01  DBKEY                      PIC S9(08) COMP-1.
 01  CHAVE-WS                   PIC S9(08) COMP-1.
*
 01  REGISTRO-SUPD-FORM.
     05 SN                      PIC  9(05) COMP-1.
     05 SNAME                   PIC  X(30).
     05 SCITY                   PIC  X(15).
*
 01  REGISTRO-PART-FORM.
     05 PN                      PIC  9(05) COMP-1.
     05 PNAME                   PIC  X(30).
     05 COLOR                   PIC  X(10).
     05 WEIGHT                  PIC  9(05) COMP-1.
*
 01  REGISTRO-SUPM-FORM.
     05 SSN                     PIC  9(05) COMP-1.
     05 SPN                     PIC  9(05) COMP-1.
     05 QTY                     PIC  9(05) COMP-1.
*
 01  LINHA-WS.
     05 FILLER                  PIC X(05) VALUE SPACES.
     05 REGISTRO-PRT            PIC X(51).
     05 FILLER                  PIC X(02) VALUE SPACES.
     05 DBKEY-PRT               PIC 9(08).
*
 01  LINHA-CAB-1.
     05 FILLER                  PIC X(20) VALUE SPACES.
     05 FILLER                  PIC X(53) VALUE
        "PROGRAMA CARGABD - CARGA DE BANCO DE DADOS".
     05 FILLER                  PIC X(05) VALUE "PAG.".
     05 CONT-PAG                PIC 99 VALUE 0.
*
 01  LINHA-CAB-2.
     05 FILLER                  PIC X(5) VALUE SPACES.
     05 FILLER                  PIC X(51) VALUE
        "REGISTRO".
     05 FILLER                  PIC X(10) VALUE "  DBKEY".
*
 01  LINHA-CAB-3.
```

```
      05 FILLER                    PIC X(5) VALUE SPACES.
      05 FILLER                    PIC X(51) VALUE ALL '-'.
      05 FILLER                    PIC X(2) VALUE SPACES.
      05 FILLER                    PIC X(6) VALUE ALL '-'.
*
 PROCEDURE DIVISION.
 INICIO.
      OPEN INPUT ENTRADA OUTPUT SAIDA.
      CALL 'OPEN' USING NUMPAG, MODO, RET-CODE.
      READ ENTRADA AT END MOVE 9 TO CHAVE-FIM.
*
 CORPO.
      PERFORM CARREGA UNTIL EOF.
*
 TERMINO.
      CALL 'CLOS' USING RET-CODE.
      CLOSE ENTRADA SAIDA.
      STOP RUN.
*
 CARREGA.
      IF TIPO = 1
      THEN PERFORM CARREGA-SUPO
      ELSE IF TIPO = 2
           THEN PERFORM CARREGA-PART
           ELSE PERFORM CARREGA-SUPM.
      PERFORM IMPRESSAO.
*
 CARREGA-SUPO.
      MOVE CORR LAY-OUT-SUPO TO REGISTRO-SUPO-FORM
      MOVE 'SUPO' TO NOMREG
      CALL 'CRS' USING NOMREG DBKEY REGISTRO-SUPO-FORM
      MOVE '$SNUM' TO NOMSET
      CALL 'AMS' USING NOMSET NOMREG RET-CODE
      MOVE '$SNAME' TO NOMSET
      CALL 'AMS' USING NOMSET NOMREG RET-CODE.
*
 CARREGA-PART.
      MOVE CORR LAY-OUT-PART TO REGISTRO-PART-FORM
      MOVE 'PART' TO NOMREG
      CALL 'CRS' USING NOMREG DBKEY REGISTRO-PART-FORM
      MOVE '$PNUM' TO NOMSET
      CALL 'AMS' USING NOMSET NOMREG RET-CODE
      MOVE '$PNAME' TO NOMSET
      CALL 'AMS' USING NOMSET NOMREG RET-CODE
*
 CARREGA-SUPM.
      MOVE CORR LAY-OUT-SUPM TO REGISTRO-SUPM-FORM
      MOVE 'SUPM' TO NOMREG
      CALL 'CRS' USING NOMREG DBKEY REGISTRO-SUPM-FORM
      MOVE '$SUPM' TO NOMSET
      CALL 'AMS' USING NOMSET NOMREG RET-CODE
      MOVE '$SNUM' TO NOMSET
      MOVE SSN OF REGISTRO-SUPM-FORM TO CHAVE-WS
      CALL 'FMSK' USING NOMSET CHAVE-WS RET-CODE
      MOVE 'SD$SM' TO NOMSET-1
```

```
           CALL *SOM* USING NOMSET-1 NOMSET RET-CODE
           CALL *AMS* USING NOMSET-1 NOMREG RET-CODE
           MOVE *$PNUM* TO NOMSET
           MOVE SPN OF REGISTRO-SUPM-FORM TO CHAVE-*S
           CALL *FMSK* USING NOMSET CHAVE-*S RET-CODE
           MOVE *PA$SM* TO NOMSET-1
           CALL *SOM* USING NOMSET-1 NOMSET RET-CODE
           CALL *AMS* USING NOMSET-1 NOMREG RET-CODE
      *
       IMPRESSAO.
           IF CONT-LIN > 55
           THEN PERFORM CABECALHO.
           MOVE DBKEY TO DBKEY-PRT.
           MOVE REG-ENTRADA TO REGISTRO-PRT.
           WRITE LINHA FROM LINHA-*S AFTER ADVANCING 2 LINES.
           ADD 2 TO CONT-LIN.
           READ ENTRADA AT END MOVE 9 TO CHAVE-FIM.
      *
       CABECALHO.
           ADD 1 TO CONT-PAG.
           WRITE LINHA FROM LINHA-CAB-1 AFTER ADVANCING SALTO.
           WRITE LINHA FROM LINHA-CAB-2 AFTER ADVANCING 2 LINES.
           WRITE LINHA FROM LINHA-CAB-3 AFTER ADVANCING 1 LINES.
           MOVE 0 TO CONT-LIN.
*FOR
```

EXEMPLO 1: LISTAGEM PRODUZIDA

PROGRAMA CARGABD - CARGA DO BANCO DE DADOS

REGISTRO		DBKEY
100001SUPRIDOR NUMERO UM	CIDADE SUPRID 1	00020496
100002SUPRIDOR NUMERO DOIS	CIDADE SUPRID 2	00020495
100003SUPRIDOR NUMERO TRES	CIDADE SUPRID 3	00020494
100004SUPRIDOR NUMERO QUATRO	CIDADE SUPRID 4	00020493
100005SUPRIDOR NUMERO CINCO	CIDADE SUPRID 5	00020492
100006SUPRIDOR NUMERO SEIS	CIDADE SUPRID 6	00020491
100007SUPRIDOR NUMERO SETE	CIDADE SUPRID 7	00020490
100008SUPRIDOR NUMERO OITO	CIDADE SUPRID 8	00020489
100009SUPRIDOR NUMERO NOVE	CIDADE SUPRID 9	00020488
100010SUPRIDOR NUMERO DEZ	CIDADE SUPRID10	00020487
210001PARTE NUMERO UM	AMARELA 00010	00020486
210002PARTE NUMERO DOIS	VERMELHA 00020	00020485
210003PARTE NUMERO TRES	VERDE 00015	00020484
210004PARTE NUMERO QUATRO	VERMELHA 00005	00020483
210005PARTE NUMERO CINCO	BRANCA 00030	00020482
3000081000100010		00020481
3000081000300020		00020480
3000091000400030		00020479
3000081000500040		0003050E
3000091000200010		00030503
3000091000400020		00030504
3000101000100010		00030503
3000111000300010		00030502

3000051000400010	00030301
3000051000500020	00030300
3000061000200010	00030499
3000061000400020	00030498
3000031000200010	00030447

PROGRAMA CARGABD - CARGA DO BANCO DE DADOS

REGISTRO	DBKEY
3000031000200020	00030449

EXEMPLO 2: LISTAGEM DOS SUPRIDORES E RESPECTIVAS CHAVES

```
LISTSD.
USER,USUARIO,SENHA.
CHARGE,CONTA,USUARIO.
ATTACH,DBMSLIB/UN=KAPINF5.
ATTACH,BANCO.
LIBRARY,DBMSLIB.
FTN5.
LGO.
*EOR
      PROGRAM LISTSD(OUTPUT,TAPE6=OUTPUT)
C     *---------------------------------------------------------------*
C     * FUNCAO: LISTAR OS REG. DE SUPRIDORES E RESPECTIVAS DBKEYS     *
C     *---------------------------------------------------------------*
      IMPLICIT INTEGER (A-Z)
      CHARACTER NOMSET*10,ENTPNT*10,SNAME*30,SCITY*15
      CALL OPEN(3,'READ',RETCOD)
      IF (RETCOD.NE.0) STOP
      CTLIN = 60
      CTPAG = 0
      CALL FFM('$SNAME',RETCOD)
    1 CALL GKM('$SNAME',DBKEY,RETCOD)
      CALL GFM('SNUM','$SNAME',SNUM,RETCOD)
      CALL GFM('SNAME','$SNAME',SNAME,RETCOD)
      CALL GFM('SCITY','$SNAME',SCITY,RETCOD)
      CTLIN = CTLIN + 1
      IF (CTLIN.GT.55) CALL CABEC(CTLIN,CTPAG)
      WRITE(6,820) SNUM,SNAME,SCITY,DBKEY
      CALL FNM('$SNAME',RETCOD)
      IF (RETCOD.EQ.0) GOTO 1
      CALL CLOS(RETCOD)
      STOP
  820 FORMAT(2X,I5,1X,A30,1X,A15,1X,I8)
      END

      SUBROUTINE CABEC(CTLIN,CTPAG)
      IMPLICIT INTEGER(A-Z)
      CTLIN = 6
      CTPAG = CTPAG + 1
      WRITE(6,10) CTPAG
      RETURN
   10 FORMAT('1',16X,'LISTSD - REGISTROS RECUPERADOS',15X,'PAG. ',I3,
     1       ///,2X,'SN',2X,'SNAME',26X,'SCITY',11X,'DBKEY',
     2       /,2X,5('-'),1X,30('-'),1X,15('-'),1X,8('-'),/)
      END
```

EXEMPLO 2: LISTAGEM PRODUZIDA

LISTSD - REGISTROS RECUPERADOS PAG.

SN	SNAME	SCITY	DEALY
5	SUPRIDOR NUMERO CINCO	CIDADE SUPRID 5	20492
10	SUPRIDOR NUMERO DEZ	CIDADE SUPRID10	20497
2	SUPRIDOR NUMERO DOIS	CIDADE SUPRID 2	20495
9	SUPRIDOR NUMERO NOVE	CIDADE SUPRID 9	20499
8	SUPRIDOR NUMERO OITO	CIDADE SUPRID 8	20498
4	SUPRIDOR NUMERO QUATRO	CIDADE SUPRID 4	20493
6	SUPRIDOR NUMERO SEIS	CIDADE SUPRID 6	20491
7	SUPRIDOR NUMERO SETE	CIDADE SUPRID 7	20496
3	SUPRIDOR NUMERO TRES	CIDADE SUPRID 3	20494
1	SUPRIDOR NUMERO UM	CIDADE SUPRID 1	20498

EXEMPLO 3: LISTAGEM DOS SUPRIMENTOS DE UM SUPRIDOR

```
LISTSP.
USER,USUARIO,SENHA.
CHARGE,CONTA,USUARIO.
ATTACH,DBMSLIB/UN=KAPINF5.
ATTACH,BANCO/UN=KAPINF5.
LIBRARY,DBMSLIB.
FTN5.
LGO.
*EOR
      PROGRAM LISTSP(OUTPUT,TAPE6=OUTPUT)
C     *---------------------------------------------------------------*
C     * FUNCAO: LISTAR TODOS OS SUPRIMENTOS DO SUPRIDOR 00005       *
C     *---------------------------------------------------------------*
      IMPLICIT INTEGER (A-Z)
      CHARACTER NOMSET*10,ENTPNT*10,SNAME*30,PNAME*30
      CALL OPEN(5,↑READ↑,RETCOD)
      IF (RETCOD.NE.0) STOP
      CTLIN = 60
      CTPAG = 0
      CALL FMSK(↑$SNUM↑,00005,RETCOD)
      CALL GFM(↑SNAME↑,↑$SNUM↑,SNAME,RETCOD)
      CALL SOM(↑SD$SM↑,↑$SNUM↑,RETCOD)
      CALL FFM(↑SD$SM↑,RETCOD)
    1 IF (RETCOD.NE.0) GOTO 2
      CALL GFM(↑QTY↑,↑SD$SM↑,QTY,RETCOD)
      CALL SMM(↑PA$SM↑,↑SD$SM↑,RETCOD)
      CALL GFO(↑PNAME↑,↑PA$SM↑,PNAME,RETCOD)
      IF (CTLIN.GT.55) CALL CABEC(CTLIN,CTPAG,SNAME)
      WRITE(6,820) PNAME,QTY
      CALL FNM(↑SD$SM↑,RETCOD)
      GOTO 1
  2   CALL CLOS(RETCOD)
      STOP
 820  FORMAT(20X,A30,2X,I5)
      END

      SUBROUTINE CABEC(CTLIN,CTPAG,PNAME)
      IMPLICIT INTEGER(A-Z)
      CHARACTER*30 PNAME
      CTLIN = 6
      CTPAG = CTPAG + 1
      WRITE(6,10) CTPAG,PNAME
      RETURN
 10   FORMAT(↑1↑,16X,↑LISTSP - LISTAGEM DOS SUPRIMENTOS↑,12X,↑PAG. ↑,I3,
     1       //,10X,↑SUPRIDOR:↑,A30,//,20X,↑PECA↑,28X,↑QTY.↑,
     2       /,20X,30(↑-↑),2X,5(↑-↑),/)
      END
```

EXEMPLO 3: LISTAGEM PRODUZIDA

LISTSP - LISTAGEM DOS SUPRIMENTOS PAG. 1.

SUPRIDOR:SUPRIDOR NUMERO CINCO

PECA	QTY.
PARTE NUMERO QUATRO	10
PARTE NUMERO CINCO	20