Impl.

UNIVERSIDADE ESTADUAL DE CAMPINAS

FACULDADE DE ENGENHARIA ELÉTRICA

DEPARTAMENTO DE COMPUTAÇÃO E AUTOMAÇÃO INDUSTRIAL

Este exemplar corresponde a redação final da tese defendida por Sergio Quezada Gonzalez e aprovada pela comissão julgadora em 16/1/1987

L. P. Magalhães

LIGG

UMA LINGUAGEM GRÁFICA BASEADA EM GRAFO PARA O MER/PAC

Por : Prof. Sergio Quezada Gonzalez

Orientador: Prof. Dr. Léo Pini Magalhães

/87

Tese de Mestrado apresentada à Faculdade de Engenharia Elétrica da Universidade Estadual de Campinas.

JANEIRO DE 1987.

Este trabalho contou com o apoio financeiro dos seguintes órgãos:

Universidade de Santiago do Chile

- USACH -

Conselho Nacional de Desenvolvimento Científico e Tecnológico

- CNPQ -

dedico este trabalho à

María Verónica

Sergio Andrés

Rodrigo Patricio

AGRADECIMENTOS

Ao professor Léo Pini Magalhães, pela amizade, confiança, incentivo e valiosa orientação, elementos que contribuiram grandemente na conclusão do presente trabalho.

Aos colegas Helga, Armando e Ivan, por sua valiosa e sempre oportuna ajuda, bem como pela amizade dispensada.

Aos professores Mario Jino e Clesio Luis Tozzi pelas correções e sugestões.

A Universidade de Santiago do Chile e CNPQ pelo apoio financeiro.

Aos colegas María Nuria, Edvaldo, Heraldo e Artemio pela convivência agradável neste período.

A Maggi e Juán por sua confiança e apoio.

A María Verónica, Sergio Andrés e Rodrigo Patricio pelo apoio nos momentos difíceis e pela compreensão pelos momentos que não pude participar de nossa convivência familiar.

E a todos aqueles que colaboraram, direta ou indiretamente, para a realização deste trabalho.

RESUMO

Neste trabalho uma linguagem gráfica baseada em grafos (LIGG) para o MER/PAC é proposta.

A definição da LIGG é feita através de uma linguagem de especificação de projetos que possui estrutura de controle semelhante às oferecidas em linguagens de alto nível.

A LIGG permite a definição e manipulação de uma base de dados no contexto MER/PAC.

O sistema de diálogo que a LIGG oferece como instrumento de comunicação ao usuário, após procedimentos de consistência no comando de diálogo, comunica-se com o GERPAC que através do SIGA acessa a base de dados.

Os procedimentos que definem a LIGG são apresentados e um exemplo de implementação é desenvolvido.

ABSTRACT

In this work, a graphical language based on graphs (LIGG) for the MER/PAC is proposed.

The definition of LIGG is done through a project specification language with a control structure similar to that from high level languages.

With LIGG one can define and manipulate a MER/PAC data base.

The dialog system supported by LIGG for user communication, after consistency procedures, use the DBMS GERPAC functions that access the data base through the kernel system SIGA.

The procedures that define LIGG are presented and a prototype was developed.

INDICE:

1. INTRODUÇÃO

O presente trabalho de tese tem sua origem no modelo de dados Entidade Relacionamento proposto por PETER CHEN em 1976. A partir de uma extensão deste modelo voltada para aplicações PAC (MER/PAC), pretende-se definir uma linguagem baseada em conceitos gráficos para o tratamento da informação.

O trabalho enquadra-se em um contexto maior na área de Projeto Auxiliado por Computador/Banco de Dados (PAC/BD), atualmente em desenvolvimento na FEE/UNICAMP, incluindo os seguintes aspectos:

a. Desenvolvimento do sistema SIGA (/RIC 86/); o qual é um sistema básico de gerência de dados que tem por principal função servir como suporte a Sistemas de Gerenciamento de Banco de Dados (SGBD), oferecendo a estes uma interface a nível lógico.

Este sistema foi desenvolvido pelo Grupo PAC/BD-UNICAMP, tendo sido sua especificação baseada no Sistema CORAS-UNICAMP atualmente implementado no mesmo local. As modificações propostas em relação ao sistema anterior visam principalmente oferecer a flexibilidade necessária a sua utilização em aplicações PAC.

Uma outra característica do SIGA é a capacidade de controlar diversas bases de dados, onde cada uma delas constitui um Sistema de Arquivos. Esta é uma das necessidades básicas em aplicações PAC, onde é necessário haver o gerenciamento de informações a nível global e a nível local simultaneamente (controle de macro-aspecto e micro-aspecto).

b. Extensão do MER para aplicações PAC - MER/PAC (/DELGA 87/): trabalho cujo objetivo é apresentar um modelo de dados aplicável a PAC, através da definição de primitivas de modelagem que permitam a formalização de todos os requisitos impostos por PAC. O modelo proposto é denominado MODELO ENTIDADE-RELACIONAMENTO PARA PAC (MER/PAC). A estrutura do modelo baseia-se essencialmente nos conceitos já definidos no Modelo Entidade-Relacionamento proposto em (/CHEN 76/), com extensões e modificações apropriadas ao tratamento dos aspectos micro e macro que caracterizam um processo de projeto PAC.

c. De forma a oferecer uma interface não-procedural para o GERPAC (Sistema Gerenciador para Projeto Auxiliado por Computador), definiu-se o presente trabalho, onde a LIGG (Linguagem Gráfica baseada em Grafo) será apresentada.

Vários fatores foram considerados na construção de tal Linguagem. Por exemplo, a linguagem deverá prover as facilidades necessárias para que o usuário possa criar e manipular visões individuais e globais.

Como é suposta a utilização da linguagem como uma ferramenta no processo de projeto, a interface do usuário deverá ser amigável e flexível. Estes aspectos são importantes porque é esperado que durante a evolução do esquema, uma grande quantidade de modificações poderão ser feitas. A característica dinâmica é uma consequência direta da aplicação.

Uma outra consideração importante é que as operações providas deverão permitir que o usuário possa definir qualquer objeto facilmente, mesmo em se tratando de um usuário eventual, que não esteja familiarizado com técnicas gráficas interativas.

A nível de usuário, a interface gráfica interagirá com o usuário através de grafos e, apoiando-se em estruturas internas e após procedimentos de consistência global, terá seus comandos traduzidos internamente para chamadas a funções GERPAC.

No capítulo 2 é apresentado um resumo sobre os modelos de dados mais conhecidos, especificamente os modelos Rede, Relacional, Conjunto Entidade e Entidade Relacionamento, assinalando suas principais características, vantagens e desvantagens. Também descreve-se uma proposta para atender as características necessárias ao modelagem de dados em Projetos Auxiliados por Computador (MER/PAC).

No capítulo 3 discutem-se as principais aproximações de linguagens de manipulação de dados, mostrando através de exemplos as linguagens de manipulação mais conhecidas.

No capítulo 4 analisa-se a computação gráfica como elemento de apoio a interação e apresentam-se as especificações da LIGG.

No capítulo 5 apresenta-se a implementação de um subconjunto de operações no MER/PAC utilizando a LIGG.

O capítulo 6 contém as conclusões do presente trabalho.

O capítulo 7 contém bibliografia de suporte ao trabalho.

Os anexos contêm o detalhamento do exemplo e das especificações e procedimentos da LIGG.

2. MODELAMENTO DE DADOS EM PAC

2.1 INTRODUÇÃO

A primeira decisão importante para a implementação de um projeto de base de dados é a escolha do modelo de dados. Claramente cada modelo de dados apresenta certas vantagens e desvantagens dependendo do ponto de vista e objetivo do projetista da base de dados. Como a escolha do modelo afeta radicalmente a natureza de uma base de dados, deve ser feita com muito cuidado. As características principais do modelo de dados consideradas para uma implementação são (/PHILIPS 84/) entre outras:

1. O modelo deverá refletir a estrutura dos dados de forma tal que os usuários da base de dados possam reconhecer o mapeamento do mundo real na base de dados. Isto é, o modelo deverá conter de preferência conceitos familiares ao ser humano refletindo as estruturas armazenadas no computador.

2. O modelo deverá facilitar a verificação da integridade dos dados. Se tal verificação de integridade é provida, o modelo poderá até representar informação semântica extra acerca do dado.

3. O modelo deverá ser suficientemente simples para tornar possível sua implementação. Claramente uma implementação que, ante consultas, tenha tempos de respostas muito pobres não é recomendável. Este requerimento, entretanto, tem que ser balanceado com o objetivo (1), em que os sistemas mais facilmente compreensíveis pelo homem têm como consequência (atualmente) uma lentidão maior na sua operação. Consequentemente o sistema requerido não é o sistema mais rápido possível, senão o que opera a uma velocidade aceitável.

4. O modelo deverá prover um sistema de consulta interativa.

Uma outra consideração importante na escolha do modelo de dados é o nível de abstração dos dados que ele oferece. Ainda que não exista uma padronização na classificação destes níveis, geralmente são quatro os considerados:

. nível conceitual;

. nível descritivo;

. nível lógico;

. nível físico.

Os níveis conceitual e descritivo referem-se à visão do mundo real, enquanto os níveis lógico e físico estão mais próximos da concepção do mundo computacional. Através de uma sequência de procedimentos conhecida como **mapeamento**, as informações modeladas na mente dos usuários (nível conceitual) serão transformadas em dados físicos armazenados nas memórias físicas dos computadores (nível físico).

Em termos dos critérios assinalados quatro modelos foram considerados e avaliados, como apresentado a seguir.

2.2 MODELOS DE DADOS USUAIS

2.2.1 MODELO REDE

A crítica principal a este modelo deve-se ao fato de não refletir adequadamente o significado dos dados. Uma vez que relacionamentos são definidos separados de entidades, não existem meios de registrar as informações sobre esses relacionamentos no modelo rede. A caracterização de propriedades de integridade não estão presentes neste modelo e como regra geral são agregadas nas implementações particulares. Assim por exemplo a caracterização de integridade na implementação CODASYL pode ser diferente da caracterização de integridade em outra base de dados usando o modelo rede.

O modelo rede não provê um sistema de consulta on-line particularmente fácil. Diferentemente do modelo relacional não existe uma base matemática para a linguagem de consulta.

Dentre os modelos aqui apresentados é o que mais se aproxima do mundo computacional. A proposta clássica do modelo rede originou-se do relatório do grupo CODASYL do DBTG, em 1971.

No modelo rede, distinguem-se duas primitivas: **tipos de registros e tipos de conjuntos.** No mapeamento do nível descritivo ao nível lógico, os tipos de registros são utilizados para representar os tipos de entidades, definidos pelos usuários e os tipos de conjuntos para representar as associações entre os tipos de entidades.

Os tipos de registros são caracterizados pelos **itens de dados.** Os valores (ocorrência dos itens de dados) assumidos pelos itens definem unidades distintas, denominadas **ocorrências de registro**, de um mesmo tipo de registro. Um grupo de valores de itens que define univocamente ocorrências de um tipo de registro é designado **chave** do tipo de registro.

No modelo rede, a correspondência entre dois tipos de registros distintos (fx: x-->y) é denominada **tipo de conjunto**, onde X é considerado **tipo de registro proprietário** e Y, **tipo de registro membro.** A associação definida no modelo rede representa a abstração da conexão física existente entre os dados na base de dados. Para obter os registros desejados, o usuário é obrigado a seguir os caminhos estabelecidos entre os registros. Este tipo de acesso é conhecido como **navegação** e nele se seguem explicitamente os caminhos de acesso para obter as informações desejadas.

Cada associação fxi, entre uma ocorrência xi do tipo de registro proprietário X e um conjunto de ocorrências do tipo de registro-membro Y, yi (yi $\geq$ 0 elementos) é designada uma **ocorrência do conjunto**, devendo satisfazer as seguintes regras de correspondência:(/TAYL 76/)

. dado um registro proprietário, é possível acessar os registros-membros;

. dado um registro-membro, é possível obter o seu registro-proprietário;

. dado um registro-membro, é possível obter os outros registros-membros pertencentes à mesma ocorrência do tipo de conjunto;

. cada registro membro só pode estar associado à um registro-proprietário em um tipo de conjunto.

O modelo rede suporta os três tipos de relacionamentos: 1:1, 1:N; M:N. Pode-se observar que o relacionamento M:N entre registros de diferentes tipos de registros violaria a quarta regra, pois um registro-membro poderia estar associado a diferentes registros-propietários. Isto é contornado no modelo CODASYL, definindo-se um terceiro tipo de registro o **pseudo registro** ou **tipo de registro relacionamento** . Com este tipo de registro, é possível decompor um relacionamento original M:N em dois relacionamentos 1:M e 1:N (/TAYL 76/).

Como consequência da proximidade do modelo rede do mundo computacional, nas suas linguagens de dados são incluídos comandos referentes aos detalhes de acesso que foram estabelecidos entre os registros para obter as informações desejadas, tais como: FIND NEXT, MOVE, GET NEXT.

O usuário também poderá intervir na organização física e endereçamento lógico dos dados, através de comandos específicos da **linguagem de descrição de armazenamento** (LDA), tais como: LOCATION MODE, ORDER, ÁREA, etc. Para assegurar a privacidade dos dados de cada usuário, a linguagem provê o comando PRIVACY, o qual deve ser declarado junto com os outros comandos da LDD, de modo que os dados protegidos só poderão ser manipulados, quando for fornecido o código PRIVACY.

Com a LDD, o administrador do banco de dados define e declara todos os tipos de dados que comporão a base de dados.

2.2.2 MODELO RELACIONAL

Implementações deste modelo tem chegado a ser mais comuns recentemente, apesar das dificuldades iniciais para se produzir um sistema de base de dados relacional eficiente. Vários sistemas comerciais estão correntemente disponíveis para uma variedade de máquinas. O modelo relacional oferece as vantagens de uma estrutura matemática, a qual permite a decomposição de consultas.O modelo relacional é um modelo não-navegacional, em oposição ao modelo rede. Decomposição de consultas no sistema relacional têm sido implementadas em diferentes formas (ex. /CHAMBERLIN 74/,/BOYCE 74/).

O modelo relacional não faz provisão para a manutenção de informação semântica adicional na descrição do dado na base de dados. Isto permite que o usuário de uma base de dados relacional confunda uma entidade com um relacionamento. Isto pode conduzir a resultados inválidos sem que o usuário fique sabendo da inconsistência.

Não existe um mecanismo de construção no modelo relacional para garantir que os dados armazenados na base de dados estejam dentro de limites razoáveis (/PHILIPS 84/).

O modelo relacional é um modelo baseado na teoria matemática de relações para representar as associações tanto entre os atributos das entidades de um tipo de entidades como entre os diferentes tipos de entidades.

Matematicamente, as relações podem ser definidas como segue (/DATE 74/).

Dado um conjunto de domínios C1, C2,...,Cn não necessariamente distintos, define-se como uma RELAÇÃO R sobre os n conjuntos, o conjunto de n tuplas, <d1,d2,...,dn>, onde d1 $\in$ C1, d2 $\in$ C2,...dn $\in$ Cn.

Fazendo uma analogia com o modelo rede, uma tupla corresponderia a uma ocorrência de um tipo de registro e uma relação corresponderia tanto a um tipo de registro como a um tipo de conjunto. Enquanto no modelo rede faz-se a distinção entre as entidades(registros) e os elos que as interligam logicamente na base de dados, no modelo relacional todas as associações são representadas indistintamente como relações.

Cada RELAÇÃO é um conjunto de tuplas definidas em um conjunto de "n" domínios C1, C2,...,Cn, não necessariamente distintos. A quantidade "n" define o grau da relação R e a quantidade de tuplas define a sua CARDINALIDADE (/CHAM 76/). Os valores que definem cada elemento de uma tupla são chamados COMPONENTES da tupla. Um conjunto de componentes referentes a uma tupla define uma OCORRÊNCIA.

Graficamente, uma relação R n-ária pode ser representada como uma tabela (/CODD 70/) onde:

. cada linha representa uma tupla da relação;

. a ordem das linhas é irrelevante;

. todas as linhas são distintas, ou seja, pelo menos um componente de uma linha é diferente dos componentes das outras linhas;

. a quantidade de colunas representa o grau da relação;

. a quantidade de linhas representa a cardinalidade da relação;

. cada coluna corresponde a um domínio da relação.

Distinguem-se dois tipos de relações: Relação de Domínios Ordenados e Relação de Domínios não Ordenados. No primeiro caso, a ordem das colunas é relevante, pois corresponde à ordem dos conjuntos C1, C2,...Cn, sobre os quais a relação R é definida. Entretanto, se cada coluna for referida pelo nome do domínio e não pela sua posição relativa dentro da relação, a ordem das colunas passa a ser irrelevante.

A coluna ou conjunto de colunas (domínios), cujos valores identificam univocamente cada elemento (tupla) em uma relação é denominada CHAVE-CANDIDATA. É possível que uma relação tenha duas ou mais chaves-candidatas. Neste caso, seleciona-se arbitrariamente uma destas chaves como a CHAVE-PRIMARIA, ou simplesmente CHAVE, da relação (/CODD 70/).

No modelo relacional não há declaração explícita das associações entre diferentes relações (existe só uma primitiva: relação). A referência das tuplas de uma relação às tuplas de uma outra relação é feita através de REFERÊNCIA CRUZADA, usando uma CHAVE ESTRANGEIRA (Foreign Key).

Denomina-se uma chave, CHAVE ESTRANGEIRA da relação R na referência à relação S, se ela não for a chave primária da relação R, mas seus elementos são valores da chave primária da relação S, de tal sorte que, a partir dela pode-se acessar os valores dos atributos definidos na relação S, associando o elemento da relação R ao elemento da relação S.

O modelo relacional é um modelo que apresenta um grau de abstração de dados maior que o modelo rede, pois nele é suposto que os ABDs (Administradores de Banco de dados) não precisam ter conhecimento dos detalhes de endereçamento lógico dos dados para acessá-los. Assim, a sua linguagem de dados é menos procedural que a do modelo rede e muitos autores preferem classificá-la como query language. Destacam-se três tipos clássicos de representação desta linguagem:

. Álgebra relacional: consiste em uma coleção de operadores que manipulam as relações para criar novas relações, isto é, as tuplas buscadas podem ser consideradas como uma nova relação que resulta de uma sequência ordenada de operações em um conjunto de relações já existentes. Os principais operadores da álgebra relacional são: PROJEÇÃO, JUNÇÃO, PERMUTAÇÃO, UNIÃO, COMPOSIÇÃO e RESTRIÇÃO (/CODD 70/).

. Cálculo Relacional sobre tuplas: a operação entre as tuplas armazenadas na base de dados faz-se através da especificação das qualificações (valores dos atributos) que estas tuplas devem satisfazer, utilizando operadores , tais como: $<$, $<=$, $>$, $>=$, $=$, $\neq$, $\exists$, $\forall$, etc. Deste modo, as linguagens baseadas em cálculo relacional apresentam um nível de abstração maior que as de álgebra relacional, Já que, na linguagem de álgebra relacional, a ordem das operações a serem efetuadas para obter uma informação é definida pelos usuários e na linguagem de cálculo relacional, isso fica por conta de um compilador ou interpretador.

. Cálculo relacional sobre domínio: aplicam-se os operadores condicionais sobre os domínios, em que são definidas as relações, ao invés de serem aplicados sobre as tuplas. Há implementações desta linguagem em forma gráfica, onde as relações são representadas no display como tabelas, os usuários preenchem os valores conhecidos e marcam os domínios e esperam que os valores desejados sejam retornados. Exemplo disto é Query-by-Example apresentado posteriormente.

2.2.3 MODELO CONJUNTO-ENTIDADE

O modelo conjunto-entidade como proposto por (/SENKO 73/) e implementado em DIAM aparece para resolver o problema de integridade de dados apresentado no modelo relacional.

Diferentemente do modelo relacional, o modelo conjunto-entidade faz diferença entre entidades e relacionamentos, entretanto informação acerca de relacionamentos não pode ser armazenada.

Existem duas críticas práticas ao modelo conjunto-entidade as quais o fazem menos que satisfatório para ser usado em um projeto de base de dados (/PHILIPS 84/).

Primeiro, algumas implementações deste modelo de dados provam a dificuldade de seu uso em ambientes comerciais.

Segundo, o modelo conjunto-entidade não provê uma base matemática para uma linguagem de consulta similar à álgebra relacional no modelo relacional. Não obstante o sistema DIAM é capaz de responder consultas on-line, embora o mecanismo pelo qual isto é executado não seja tão direto como no sistema relacional.

O elemento básico do modelo conjunto-entidade é a entidade. As entidades têm nomes. Nomes de entidades tendo propriedades em comum são agrupadas num **conjunto de nomes de entidades** o qual é referido por **nome do conjunto de nomes de entidades**, tais como NOME, COR e QUANTIDADE.

Uma entidade é representada por um par: identificação do conjunto entidade/ocorrência, como por exemplo NOME/Sergio Andrés, NUMERO-EMPREGADO/1601, e NUMERO-DE-ANOS/42. Uma entidade é descrita por sua associação com outras entidades.

Conceitualmente, o modelo conjunto-entidade difere do modelo entidade-relacionamento no seguinte:

. no modelo conjunto-entidade, cada objeto é tratado como uma entidade. Por exemplo, COR/branco e NUMERO-DE-ANOS/42 são entidades. No modelo entidade-relacionamento, branco e 42 são tratados como valores. É de observar que o tratamento de valores como entidades pode causar problemas semânticos;

. somente relacionamentos binários são usados no modelo conjunto-entidade, enquanto relacionamentos n-ários podem ser definidos no modelo entidade-relacionamento.

Uma das principais dificuldades na aprendizagem do modelo conjunto-entidade é sua visão do mundo, isto é, identificação de valores como entidades.

O modelo conjunto-entidade trata o conjunto de valores, tais como NUMERO-DE-ANOS como **conjunto de nomes de entidades** e os valores como **nomes de entidades**. Os atributos são tratados como nomes do papel no modelo conjunto-entidade.

Para relacionamentos binários, a translação é: o papel de uma entidade num relacionamento (ex: o papel de DEPARTAMENTO no relacionamento DEPARTAMENTO-EMPREGADO) é o nome do papel da entidade na descrição de outra entidade no relacionamento. Para relacionamentos n-ários, pode-se criar entidades artificiais para relacionamentos de forma a manipular os relacionamentos no mundo binário.

2.2.4 MODELO ENTIDADE-RELACIONAMENTO

O modelo entidade-relacionamento (MER) como proposto por (/CHEN 76/), foi escolhido como o modelo de dados inicial para ser usado no presente projeto.

O modelo entidade-relacionamento enquadra-se no conjunto de modelos de dados que apresentam grau de abstração correspondente ao nível descritivo. Ele dá maior ênfase à definição da semântica de informações, direcionando os usuários a definir com precisão e numa forma natural, os seus requisitos.

Conceitualmente, o relacionamento definido no modelo rede significa a conexão lógica que se estabelece entre os registros. Por sua vez, no MER, um relacionamento é mais que uma conexão lógica, ele deve ser considerado como um elemento concreto com existência própria, da mesma forma que as entidades, isto é, sem estar unicamente estabelecendo uma associação.

Um conceito importante introduzido neste modelo é a definição do relacionamento sobre um conjunto de funções, ao invés da sua definição sobre os domínios de valores, como ocorre no modelo relacional. Uma das vantagens deste conceitos é que:

. as funções desempenhadas pelas entidades num relacionamento são únicas, isto é, não há possibilidade da ocorrência de ambiguidade da interpretação de uma informação contida numa tupla.

Uma outra noção que fornece o modelo entidade-relacionamento é a classificação das entidades e relacionamentos em fracos e regulares, características que serão analisadas posteriormente.

As razões para a escolha do modelo entidade-relacionamento foram entre outras:

1. O modelo entidade relacionamento apresenta uma visão do dado a qual corresponde estritamente ao caminho no qual os dados são vistos pela mente humana. O modelo representa coleções de "coisas" e coleções de relacionamentos entre "coisas", isto é: conjunto de entidades e conjuntos de relacionamentos.

Subclassificações são também permitidas, exemplo, o conjunto entidade SAPATO pode ter subconjuntos SAPATO-ESQUERDO e SAPATO-DIREITO. Informações acerca de relacionamentos podem ser armazenadas, exemplo, o relacionamento CASADO entre duas ocorrências entidades no conjunto entidade PESSOA pode registrar NUMERO-DE-ANOS.

2. O modelo restringe a base de dados para ter informação semântica útil mantendo a integridade dos dados. Por exemplo tamanho de sapato de 900 não é permitido desde que o conjunto de valores TAMANHO-SAPATO não conteria valores acima ou abaixo dos limites razoáveis.

3. Visto que os dados numa base de dados entidade-relacionamento podem ser tratados como um conjunto de relações entidade-relacionamento, álgebra relacional pode ser usada para acessar itens de dados. Desde que este tipo de processamento de consulta é bem aprendido, a provisão de um sistema de consulta on-line para uma base de dados entidade relacionamento não deve apresentar maiores problemas.

A maior desvantagem na adoção do modelo entidade-relacionamento é a carência de informação acerca de outras implementações. Todos os outros modelos considerados têm sido implementados como sistemas de base de dados e os detalhes dessas implementações têm sido publicadas. Estes pontos serão abordados com maior profundidade a seguir.

2.3 CARACTERÍSTICAS DE APLICAÇÕES PAC

2.3.1 INTRODUÇÃO

A principal característica de aplicações PAC é a complexidade das informações envolvidas e do controle do processo de projeto e, portanto, do controle do fluxo das informações entre os diversos níveis deste processo.

A utilização de Sistemas de Banco de Dados em Sistemas PAC vem se tornando, nos últimos anos, objeto de muita pesquisa. As razões que justificam o uso de tais sistemas em PAC são entre outras:

. O fato que o sistema de banco de dados age como elemento de comunicação e integração, ligando todas as atividades PAC e fornecendo meios para a realimentação de informações entre as diversas atividades.

. Em PAC o volume de manipulações das informações é muito grande, o que reforça o uso de SGBDs para o controle da redundância, consistência e integridade dos dados, garantindo a qualidade da informação armazenada no sistema PAC.

. O sistema de banco de dados se presta como elemento de coordenação do sistema PAC, pois todas as informações pertinentes às atividades e ao fluxo de dados do sistema PAC podem ser armazenadas no banco de dados. Estas informações são atualizadas à medida que o processo de projeto evolui, o que assegura um trabalho coordenado de todos os processos residentes no sistema PAC.

. Finalmente, um sistema de banco de dados assegura aos projetistas e usuários um acesso mais rápido e eficiente a informações de seu interesse.

2.3.2 SISTEMAS DE BANCO DE DADOS PARA PAC

Usualmente, uma base de dados pode ser vista como uma coleção de dados armazenados em memória secundária, acessados por programas de aplicação batch ou por usuários on-line. Aqui, a principal noção de interesse é a de integração dos dados, isto é, a base de dados contém dados para muitos usuários, que podem acessar subconjuntos independentes da base de dados.

As vantagens mais frequentemente mencionadas da adoção de SGBD são os conceitos de: redução de redundância, avaliação de consistência, compartilhamento de dados, integridade, segurança e independência dos dados.

Vários modelos de dados têm sido providos para representar a estrutura da base de dados. Eles podem ser usados através de linguagens particulares de definição de dados (DDL) e linguagens de manipulação de dados (DML).

Dependendo da utilização do banco de dados (administração ou engenharia), pode-se considerar a existência de duas grandes linhas no desenvolvimento de software para banco de dados, conhecidas como banco de dados clássicos e de apoio a sistemas PAC cujas características e diferenças mais relevantes são:

a). em banco de dados clássicos a organização é estática, isto é, entidades e relacionamentos entre entidades não se alteram por um longo tempo. Portanto podem ser descritos e usados para fazer a carga inicial do banco de dados;

b). em PAC deve ser possível manipular duas classes de informação:

. o desenvolvimento do projeto (regras, métodos, elementos padrão, etc);

. informações sobre o objeto a ser projetado. Este grupo de dados é muito dinâmico no sentido que não são conhecidos a priori, sendo definidos ou redefinidos no transcurso do processo;

c). em banco de dados clássicos sendo a informação estática pode ser compilada;

d). em PAC, como algumas informações evoluem continuamente, o esquema deve ser dinâmico e interpretado;

e). em um projeto, existem muitas entidades e muitos relacionamentos, formando uma estrutura complexa;

f). nos banco de dados administrativos, desde que a realidade foi descrita e a base de dados carregada, o dado usado é constante e evolue lentamente na sua estrutura;

g). em projetos PAC , onde o objeto não está a-priori completamente projetado, sua descrição evolui igualmente tanto em sua estrutura como nos valores.

Da análise das características dos campos de utilização pode-se inferir que um SGBD/PAC , deve oferecer uma linguagem de definição e manipulação de dados, que ofereça ao projetista a possibilidade de novas classes de dados, ou redefinição, bem como operar sobre todos os dados armazenados ao longo do processo de projeto.

Um modelo de dados PAC deve permitir modelamentos estáticos e classes dinâmicas de entidades. Entidades estáticas pertencem a especificação do projeto e são usadas no desenvolvimento dinâmico de entidades que representam os objetos a serem projetados.

Um modelo de dados PAC, deve permitir representar objetos dinâmicos para as diferentes descrições. Como muitos projetistas trabalham sobre os mesmos objetos em seus domínios de especificação, isto leva a que alguns objetos possam ser vistos de forma diferente. Todas essas visões devem ser coerentes desde que elas modelam o mesmo objeto.

Uma outra característica importante em PAC, é a capacidade de poder especificar tipos de dados não convencionais, tais como linhas, polígonos, círculos, elementos importantes na definição de uma linguagem gráfica como a proposta no presente trabalho de tese.

2.3.3 MODELO DE DADOS MER/PAC

Vistas as principais características que devem estar presentes em SGBDs e em modelos de dados úteis a aplicações PAC, na continuação se apresenta uma proposta de um modelo de dados para aplicações PAC (/DELGA 87/), levando em consideração os aspectos anteriormente considerados.

Da análise feita pode-se inferir que o Modelo Entidade-Relacionamento é considerado um modelo adequado para a descrição de dados PAC, devido a que ele permite:

. definição de entidades (definição estrita);

. definição de relacionamentos recursivos;

. definição de relacionamentos 1:N, M:N, 1:1;

. definição de atributos multivalorados;

. definição de domínios de valores;

. explicitação da dependência de existência e de identificação entre entidades.

No entanto, com estas características o MER não satisfaz todas as exigências da aplicação PAC. Por isso foi proposta uma extensão do MER, denominada MER/PAC (DELGA 87/), com as seguintes características adicionais:

. definição complexa de objetos e associações;

. definição de relacionamentos tanto entre entidades como entre relacionamentos;

. utilização de tipos abstratos de dados ;

. explicitação de claúsulas de imposição de existência;

. explicitação de dependência de existência entre entidades e relacionamentos, ou apenas entre relacionamentos.

Assim, o MER/PAC fica definido através dos seguintes elementos:

. entidade (regular/fraca);

. relacionamento (regular/fraco);

. agrupamento (regular/fraco);

. esquema (regular/fraco);

. definição de regras.

No MER/PAC os tipos de entidades e relacionamentos são classificados de acordo com dois critérios: complexidade de definição e tipos de dependência entre as primitivas. No que se refere à complexidade, as entidades e relacionamentos suportam dois tipos de definição, já mencionados: definição estrita e definição complexa.

Quanto ao aspecto de dependência, as primitivas podem ser classificadas como regulares ou fracas, conforme seja a dependência de existência destas primitivas em relação a outras.

O que se procura frizar na proposta é que, a nível de definição, cada classe de entidade tem uma classe correspondente de relacionamentos (ex. um tipo de entidade fraca corresponde a um tipo de relacionamento fraco).

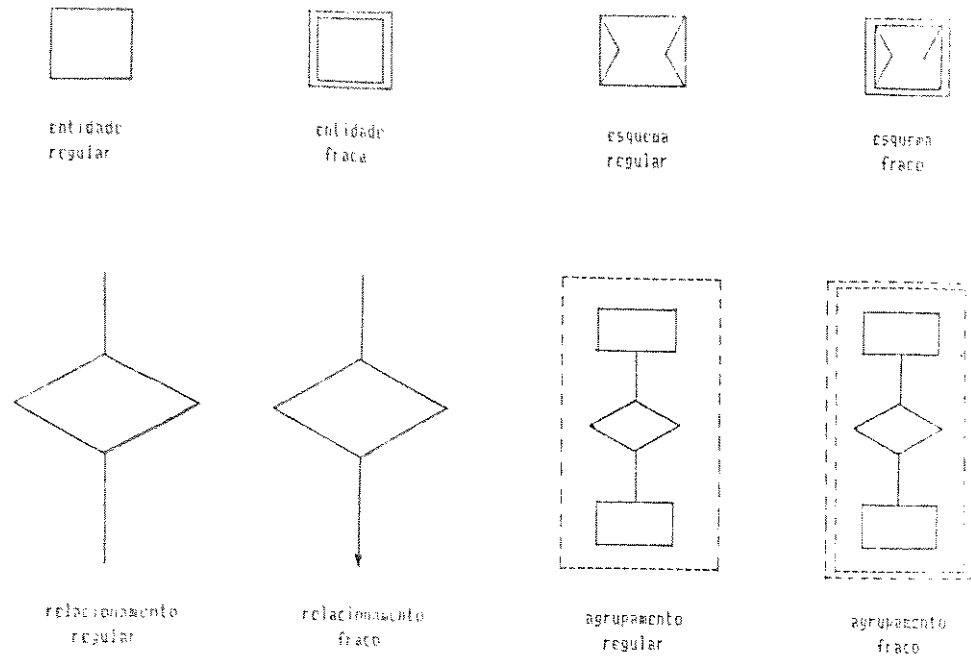


Figura 2.1 Primitivas do MER/PAC

A figura 2.1 apresenta as primitivas de um diagrama MER/PAC. Muitos autores vêm propondo novos elementos e conceitos ao Modelo Entidade-Relacionamento. Em face destas propostas, é que o trabalho de Delgado procura adaptar aqueles conceitos e elementos, propondo novas primitivas que servem aos propósitos de modelamento de PAC que são detalhados na continuação.

PRIMITIVAS DO MER/PAC

a. Entidades e Relacionamentos:

A primitiva entidade representa os objetos e fatos que têm existência e podem ser identificados no Mundo Real. As entidades que possuem propriedades comuns são classificadas em tipos de entidades.

Existem entidades que têm sua existência no banco de dados dependente da existência de outras entidades ou de um grupo de entidades, distinguindo-se em entidades regulares e fracas. Uma entidade é dita ser regular se sua existência independe da existência de outras entidades e /ou relacionamentos. Caso não tenha esta propriedade é dita fraca. Deve-se observar que o conceito de dependência de identificação (/CHEN 77/), não existe no MER/PAC. Isto porque não há no MER/PAC o conceito de chave, o que quer dizer que não há como indicar os atributos que identificam uma entidade (MER/PAC situa-se a nível descritivo).

b. Relacionamentos entre entidades e/ou relacionamentos:

No MER/PAC é possível estabelecer um relacionamento envolvendo outros relacionamentos. Isto é feito através da abstração de um ou mais relacionamentos e entidades correlatas em um novo objeto, denominado agrupamento.

A cada agrupamento deve ser associado um conjunto de regras de formação de ocorrências, que define qual o conjunto de ocorrências das entidades/relacionamentos que definem uma ocorrência do agrupamento.

Através de agrupamentos é possível definir objetos complexos e relacionamentos mutuamente exclusivos, conceitos importantes em PAC.

b.1. Relacionamentos Mutuamente Exclusivos

Relacionamentos de tipos distintos são ditos serem mutuamente exclusivos se a sua existência for mutuamente exclusiva, isto é, a ocorrência de um tipo de relacionamento implica na não existência dos outros tipos de relacionamentos.

b.2 Objetos complexos

Um objeto complexo, consiste de uma composição de objetos simples associados a um objeto raiz. Os objetos subordinados podem ser complexos ou não e, semanticamente, representam uma descrição das características do objeto raiz. A ocorrência de um objeto complexo é na realidade a ocorrência de todos os objetos que o definem.

c. Esquema:

Um esquema consiste na definição de um conjunto de primitivas e claúsulas de integridade que constituem a descrição semântica da estrutura de dados de uma determinada aplicação. Desta forma a primitiva esquema no MER/PAC representa um conjunto de entidades, relacionamentos e agrupamentos, que formam a estrutura de dados de uma determinada aplicação. Cada um destes conjuntos pode, por sua vez, sofrer alterações, tanto na estrutura de dados quanto nos dados operacionais associados. De maneira análoga a entidades , os esquemas podem ser classificados em diferentes tipos de esquemas. No MER/PAC as instâncias de um tipo de esquema são denominadas expansões, sendo que os diversos conjuntos de dados operacionais associados a uma determinada expansão denomina-se versão de uma expansão.

d. Domínio de valores:

A definição de domínio de valores se insere em dois níveis de abstração. A nível descritivo (MER/PAC), simplesmente se associa um tipo de entidade a um domínio de valores de um atributo. O domínio de valores é identificado por um nome genérico qualquer, da mesma forma que é feito para atributos e tipos de entidade e relacionamento.

A nível organizacional, o que se propõe no trabalho de Delgado é que os tipos de dados devem ser extendidos para tipos abstratos de dados (TAD), onde os tipos de dados convencionais são também considerados TADs.

e. Regras: Integridade, Consistência e Chaves

No modelo proposto estabelece-se que a cada primitiva é possível associar um conjunto de regras que deve ser aplicado durante qualquer manipulação feita sobre aquela primitiva, mantendo assim a integridade e consistência requerida pela aplicação.

Assim, no MER/PAC as regras se dividem em três grupos: regras de formação, regras de manipulação e regras estruturais. As regras de formação são aquelas que definem a forma como devem ser criadas e mantidas as ocorrências das primitivas MER/PAC. Estas regras são as utilizadas na definição de agrupamentos.

As regras de manipulação são regras que devem ser verificadas durante as operações de inserção, remoção e alteração, podendo expressar uma imposição de existência.

As regras estruturais são regras impostas sobre os valores de determinados atributos, especificando restrições quanto a valores admissíveis para o domínio de valores, valores permitidos para certos atributos, valores existentes na base de dados.

3. LINGUAGENS DE SGBDs

3.1 INTRODUÇÃO

Neste capítulo discutem-se as principais características de linguagens de manipulação de dados para efetuar consultas numa base de dados.

Várias linguagens de consulta baseadas no modelo relacional Já foram propostas, sendo que a maioria delas pode ser classificada em uma das duas categorias predominantes:

. baseada na Álgebra Relacional

. baseada no Cálculo Relacional.

Tanto a Álgebra Relacional como o Cálculo Relacional foram originalmente propostos por (/CODD 71/). A álgebra Relacional se utiliza de operações tipo "JUNÇÃO" e "PROJEÇÃO" e como foi apontado por (/CODD 71/), linguagens baseadas em Álgebra Relacional obrigam o usuário a expressar uma pergunta utilizando uma sequência de operadores, exigindo portanto um certo conhecimento de programação. Além disso o próprio usuário é obrigado a estabelecer a sequência que otimize a pesquisa a ser feita.

Já o Cálculo Relacional se utiliza de cálculo de predicados e é menos programático, no entanto, exige a utilização dos quantificadores matemáticos (existencial e universal) e de expressões booleanas complexas.

Muitos estudos continuam sendo feitos para o desenvolvimento de linguagens de consulta mais simples(naturais) e que permitam o acesso, para recuperação ou modificação, a qualquer informação contida no banco de dados. Situa-se neste grupo de linguagens a SEQUEL (/CHAMBERLIN 75/), com a introdução de palavras chaves e blocos básicos para seleção de informações. O resultado foi uma linguagem, embora um tanto programática, mais facilmente assimilável. Outra linguagem com caracteristicas semelhantes, a QUERY-BY-EXAMPLE (/ZLOOF 75/), utiliza-se de tabelas que devem ser preenchidas como exemplos de possíveis respostas.

Também é importante citar as linguagens ISBL, SQUARE e QUEL, como aportes importantes no desenvolvimento da pesquisa nesta área.

Experiências com o modelo Entidade-Relacionamento como base para linguagens de consulta são mais raras na literatura. Aqui apresentam-se dois exemplos de linguagem deste tipo, a primeira é REQUEL (/SILVA 77/) projetada para o sistema REDAS e a segunda é CLEAR (/POONEN 78/).

3.2 SEQUEL - STRUCTURED ENGLISH QUERY LANGUAGE.

SEQUEL é uma linguagem relacional usada pelo sistema R, que utiliza palavras chaves em inglês formando blocos estruturados para indicar relações e nomes de atributos. Através de exemplos, usando um banco de dados que descreve uma empresa, mostram-se os recursos oferecidos por SEQUEL. O banco de dados dos exemplos tem apenas duas tabelas:

EMP(NOME, NUM, DEPTN, CARGO, SALÁRIO, GERENTE)

DEPT(DEPTN, DNOME, LOC)

A tabela EMP contém o nome do empregado, seu número, número do seu departamento, cargo, salário e o seu gerente. A tabela DEPT contém número do departamento, nome do departamento e localização.

Uma operação simples em SEQUEL é procurar um valor associado na tabela a outro valor conhecido.

. Achar os números dos departamentos localizados em Campinas:

```
SELECT  DEPTN
FROM    DEPT
WHERE   LOC = 'CAMPINAS ';
```

Este bloco é constituído por 3 palavras chaves: SELECT, FROM e WHERE: e 3 parâmetros: a tabela DEPT, os itens a serem impressos DEPTN, e a condição que deve ser satisfeita LOC= 'CAMPINAS'.

Se a claúsula WHERE for omitida a pesquisa retornará todos os valores daquela coluna da tabela, sendo que os valores duplicados serão eliminados.

Quando se quiser imprimir toda a linha pode-se usar a abreviação SELECT *.

. Listar as linhas da tabela EMP para os empregados do Dept.63 que ganham mais de Cz 17.000.

```
SELECT *
FROM    EMP
WHERE   SALÁRIO > 17.000
AND     DEPTN  =  63
```

SEQUEL permite também fazer vários níveis de aninhamento de blocos de modo que a saída de um bloco seja a entrada de outro.

. Listar os nomes de empregados que trabalham em Departamentos localizados em Campinas.

```
SELECT  NOME
FROM    EMP
WHERE   DEPTN  IN
        SELECT  DEPTN
        FROM    DEPT
        WHERE   LOC  =  'CAMPINAS';;
```

Eventualmente pode ser necessário indicar a correlação entre dois blocos, para isso existe a variável de correlação que deve ser colocada na cláusula FROM, como exemplificado a seguir.

. Listar os nomes dos empregados que ganham mais que seus Gerentes.

```
SELECT  NOME
FROM    X  IN  EMP
WHERE   SALÁRIO  >
        SELECT  SALÁRIO
        FROM    EMP
        WHERE   NOME  =  X.GERENTE;;
```

SEQUEL possibilita ordenar dados recuperados em uma pesquisa.

. Listar nomes e salários dos empregados do Departamento 63 em ordem crescente de salário.

```
SELECT   NOME, SALÁRIO
FROM     EMP
WHERE    DEPTN  -  63
ORDER    BY  SALÁRIO  ASC;
```

Entre outros recursos da SEQUEL tem-se as claúsulas GROUP BY, HAVING e SET, as funções COUNT, SUM, AVG, MAX e MIN além das operações de atualização UPDATE, INSERT e DELETE.

3.3. QBE -QUERY-BY-EXAMPLE

Uma outra linguagem relacional, denominada QUERY-BY -EXAMPLE, foi desenvolvida pela IBM. Esta contém um número de características não presentes em Álgebra Relacional ou Cálculo Relacional ou em algumas das linguagens de consulta implementadas mais conhecidas. QBE foi desenvolvida visando usuários em terminais de vídeo e todas as operações são especificadas em tabelas, parte delas sendo preenchidas pelo próprio usuário. A princípio o sistema informa quais as tabelas disponíveis para aquele usuário e coloca uma tabela vazia na tela, o usuário então escolhe qual tabela quer pesquisar colocando o seu nome na tabela vazia. O sistema responde preenchendo o nome dos itens nas colunas. Se o usuário quiser apenas a impressão dos dados desta tabela, basta colocar "P." abaixo do nome da tabela, se quiser imprimir apenas alguns campos deve colocar "P." abaixo dos nomes dos itens desejados e, se quiser selecionar pode colocar a condição na própria tabela. Por exemplo:

EMP	NOME	NUM	CARGO	SALÁRIO	GERENTE
	P.		ANALISTA	>17.000	

Condições especificadas na mesma linha são conectadas pelo operador "E", para usar "OU" preenche-se em linhas diferentes:

EMP	NOME	NUM	CARGO	SALÁRIO	GERENTE
	P.		ANALISTA		
				>17.000	

Para fazer junção de duas tabelas usa-se um "elemento exemplo" para fazer a ligação, tendo como resultado uma terceira tabela:

EMP	NOME	NUM	CARGO	SALÁRIO	GERENTE
	SÉRGIO				RODRIGO

DEPT	DEPTN	GERENTE
	D	RODRIGO

==>

RESP	NOME	DEPTN
	P.SÉRGIO	P.D

Se a condição não couber no espaço reservado à coluna pode-se alargar o espaço para aquela coluna, colocar a condição em mais de uma linha ou preencher una "caixa de condição", por exemplo:

. Deseja-se imprimir o nome dos analistas e programadores cujo Gerente seja comum.

EMP	NOME	NUM	CARGO	SALÁRIO	GERENTE
	P.JOÃO		ANALISTA		X
	P.CARLOS		PROGRAMADOR		X

. Pesquisa idêntica a do item acima com a restrição de que o Gerente não pode ser Rodrigo.

EMP	NOME	NUM	CARGO	SALÁRIO	GERENTE
	P.JOÃO		ANALISTA		X
	P.CARLOS		PROGRAMADOR		X

CONDIÇÃO BOX
X ≠ RODRIGO

Pode-se também usar o elemento exemplo na mesma tabela:

. Quais os empregados que ganham mais que seus Gerentes.

EMP	NOME	NUM	CARGO	SALÁRIO	GERENTE
	P. X		P.	>Y Y	X

QUERY-BY-EXAMPLE possibilita alguns tipos de pesquisa de uma forma simples que, em outros linguagens seria bastante complexo:

. Recuperação do nome da tabela dado o nome da coluna.

P.	GERENTE	==>	EMP	DEPT

. Recuperação do nome da tabela e nome da coluna dado um valor.

P.	P.	==>	ENDEREÇO	CIDADE	ESTADO	CIDADE
	CAMPINAS					

QUERY-BY-EXAMPLE também possibilita fazer atualização, através dos comandos UPDATE, INSERT e DELETE e também como recurso adicional para recuperação de informações as funções COUNT e SUM.

3.4 REQUEL - RELATION ENTITY QUERY LANGUAGE

Esta linguagem foi desenvolvida utilizando-se as características de modelos baseados em classes de entidades e classes de relações. REQUEL permite tratar o conjunto de dados a um nível conceitual, onde são definidas entidades e relações (associações) entre estas entidades, o que torna a elaboração de perguntas, na maioria dos casos, mais fácil e natural.

Para a apresentação das características da linguagem, considerar-se-á um exemplo de um banco de dados de uma pequena empresa. Existem fornecedores nesta empresa, sobre os quais é importante manter-se conhecimento do número, nome e cidade onde se localizam. Existem itens nesta empresa, sobre os quais deve-se conhecer o número de série , o nome, a cor e o pêso. Os itens existentes nesta empresa são fornecidos pelos fornecedores mencionados acima, em uma determinada quantidade.

Podem-se expressar as classes apresentadas como:

```
ENTITY    F(FNUM, FNOME, CIDADE)
ENTITY    I(INUM, INOME, ICOR, IPESO)
RELATION  FI(F, I, QUANT)
```

A expressão mais simples em REQUEL é uma seleção sobre uma das classes contidas no banco de dados.

. Obter os números dos fornecedores localizados em Campinas.

```
PRINT   (F.FNUM)
WHERE   (F.CIDADE = 'CAMPINAS')
```

Pode-se recuperar valorès de todos os atributos de uma classe de entidades ou relações, bastando acrescentar após o nome da classe, um ponto e um asterisco:

. Obter todos os dados de fornecedores localizados em Campinas.

```
PRINT   (F.*)
WHERE   (F.CIDADE = 'CAMPINAS')
```

A supressão de cláusulas WHERE implica na qualificação de todas as entidades mencionadas. Por exemplo:

```
PRINT   (I.ICOR)
```

A qualificação pode conter expressões booleanas:

. Obter os números dos itens que apresentam cor vermelha ou preta e pêso maior que 20.

```
PRINT  (I.NUM)
WHERE  (I.ICOR = ('VERMELHO', 'PRETO') AND I.PESO > 20)
```

Uma classe de relações indica a associação entre duas ou mais classes de entidades e a ela podem estar associados atributos. Esta característica é fundamental no modelo adotado e deve ser preservada pela linguagem REQUEL. Não se pode obter qualquer propriedade de entidades diretamente de uma relação, mas sim utilizando esta relação.

. Obter os números dos fornecedores que fornecem item de número 18.

```
PRINT  (F.NUM)
WHERE  ((F,I) ARE RELATED BY FI AND I.INUM = 18)
```

Um bloco de seleção fornece como resultado um conjunto de valores, a linguagem permite comparações entre um elemento qualquer e este conjunto de valores obtido através dos operandos SOME e ALL.

. Obter o número dos fornecedores que tenham número igual a um dos números dos itens de cor verde.

```
Bloco      PRINT  (F.FNUM)
externo    WHERE  ((F,I) ARE RELATED BY FI AND F.FNUM=SOME)

Bloco      (I.INUM)
interno    WHERE  (I.ICOR = 'VERDE`))
```

Identificamos nesta pergunta dois blocos: um bloco interno (bloco de seleção simples) que fornece como resultado o conjunto dos números de itens de cor verde e um bloco externo que obtém valores de FNUM contidos no conjunto fornecido pelo bloco interno. Outro exemplo seria:

. Obter o nome dos fornecedores que fornecem itens cujo pêso é maior que os pêsos de todos os itens de cor vermelha.

```
PRINT  (F.FNOME)
WHERE  ((F,I)  ARE  RELATED BY FI AND  I.IPESO  >  ALL
       (I.IPESO)
       WHERE (I.ICOR = VERMELHO`))
```

A linguagem REQUEL permite a utilização de funções internas. Existem dois tipos básicos de função:

- Funções denominadas "funções simples" que atuam sobre o conjunto de valores fornecidos como argumento. São disponíveis as funções COUNT, MAX, MIN, AVG, TOTAL.

- Funções denominadas "funções seletivas". Estas funções atuam sobre conjuntos de valores de um atributo.São disponíveis as funções GROUPED BY e COUNT.

Para modificação do conteúdo do banco de dados, REQUEL tem as seguintes operações:

ADD - para adicionar uma ocorrência a uma classe de entidades.
CONNECT - para adicionar uma ocorrência a uma classe de relações.
DELETE - para retirar informações de uma classe de entidades.
DISCONNECT - para retirar informações de uma classe de relações.
UPDATE - para modificação de informações do banco de dados.

3.5 CLEAR - A CONCEITUAL LANGUAGE FOR ENTITIES AND RELATIONSHIPS

Na linguagem CLEAR, o usuário se comunica em termos das entidades e relacionamentos relevantes ao problema em questão. Além da simplicidade de uso, outra vantagem é que trabalhando no nível conceitual fica-se imune a possíveis mudanças na estrutura do banco de dados.

Para apresentar a linguagem usar-se-á como exemplo um pequeno banco de dados de uma empresa, cujo diagrama entidade-relacionamento está na figura 3.1. A vantagem da técnica de diagrama E-R é a facilidade de o usuário usar o diagrama para formular suas perguntas. Em bancos de dados grandes e complexos esta facilidade é de grande ajuda no entendimento da estrutura do banco de dados.

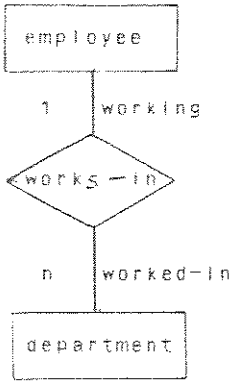


Figura 3.1 Diagrama entidade-relacionamento

A figura 3.1 descreve um banco de dados com duas entidades EMPLOYEE e DEPARTMENT onde cada empregado trabalha em apenas um departamento e cada departamento pode ter vários empregados trabalhando nele. Os arcos entre entidades e relacionamentos foram denominados WORKING e WORKED-IN (modificação da proposta de CHEN). Tanto as entidades como os relacionamentos podem ter outros atributos, a entidade EMPLOYEE tem como atributos:

NAME - nome do empregado (domínio=NOME)
EMP-NO - número do empregado (domínio=NO-EMPREGADO)
AGE - idade (domínio=IDADE)
SALARY - SALÁRIO (domínio=SALÁRIO)

e a entidade DEPARTMENT tem como atributos:

DNO - número do departamento (domínio=NO-DEPARTAMENTO)
LOCATION - localização (domínio=CIDADE)

Tratando-se de diagrama E-R existem algumas operações básicas importantes de citar. A seguir serão apresentados diversos exemplos que permitirão mostrar as características desta linguagem.

- Seleção de atributos específicos de uma entidade ou relacionamento.

. Listar o nome e o salário de todos os empregados:

```
NOME, SALÁRIO OF EMPLOYEE
```

Deve-se notar que NOME e SALÁRIO são os nomes dos domínios associados aos atributos NAME e SALARY. Quando o nome do domínio não é único em uma entidade ele deve vir acompanhado do nome do atributo. O nome da entidade pode ser usado para selecionar todos os atributos desta entidade.

- Seleção de ocorrências de uma entidade ou relacionamento que satisfazem uma determinada condição.

. Ache toda a informação sobre os empregados que ganham mais de Cz$ 22.000.

```
EMPLOYEE WITH SALARIO > 22.000
```

- Percorrendo duas ou mais entidades para obter informação baseada no relacionamento entre elas.

. Encontre a localização do departamento em que PAULO trabalha:

```
LOCATION
OF DEPARTMENT
IN WORKS-IN
WITH NOME OF EMPLOYEE = 'PAULO'
```

A frase " IN WORKS-IN " poderia ser removida se existisse apenas um relacionamento entre EMPLOYEE e DEPARTMENT, as duas entidades poderiam ter sido qualificadas como DEPARTMENT WORKED-IN e WORKING EMPLOYEE:

```
LOCATION OF DEPARTMENT WORKED-IN
WITH NOME OF WORKING EMPLOYEE = 'PAULO'
```

- Combinação entre duas entidades.

. Forme uma entidade que contenha toda a informação de um empregado e do departamento em que ele trabalha:

```
EMPLOYEE, DEPARTMENT IN WORKS-IN ou
WORKING EMPLOYEE, WORKED-IN DEPARTMENT
```

Alguns dos outros recursos fornecidos em CLEAR são BETWEEN, CONTAINS, DOES NOT CONTAIN, GROUP BY, ORDER, e as funções GREATEST, SMALLEST, SUM, MAX e MIN.

3.6 COMENTÁRIOS.

Na visão dada das diversas linguagens vários aspectos interessantes podem ser comentados, tais como:

No REQUEL as entidades são relacionadas através da frase " ARE RELATED BY " e no CLEAR através de uma palavra associada ao relacionamento.

Um aspecto interessante do REQUEL é que quando a cláusula " WHERE " for suprimida, então todas as ocorrências são qualificadas e na SEQUEL quando todos os itens de uma entidade devem ser listados não é necessario escrever o nome de todos mas apenas um " * ".

As linguagens aqui abordadas exigem um determinado grau de conhecimento de programação, apenas o QUERY-BY-EXAMPLE é mais mais facilmente assimilável por um usuário não-programador.

4. LIGG - UMA LINGUAGEM GRÁFICA BASEADA EM GRAFOS PARA O MER/PAC

4.1 COMPUTAÇÃO GRÁFICA COMO APOIO A INTERAÇÃO

Uma das formas mais efetivas de comunicação homem/máquina no contexto computacional é através da utilização de interação gráfica.

Um **sistema gráfico** pode ser definido como alguma coleção de hardware e software projetada para tornar mais fácil a entrada e saída gráfica em programas de computador. Para o usuário, os componentes relevantes ou visíveis de hardware são o dispositivo de display e uma combinação de dispositivos de entrada. O dispositivo display é usualmente um monitor. Os dispositivos de entrada são geralmente teclado alfanumérico, mouse ou light-pen.

Parte do software é um **pacote gráfico**. Isto é, um conjunto de funções ou subrotinas chamadas por um programa de aplicação para gerar figuras sobre a tela do display e para manipular interações (/ NEWMAN 81/). Um pacote gráfico consiste de diferentes conjuntos funcionais. Cada conjunto está ligado a uma classe particular de tarefa. Nos parágrafos seguintes, alguns dos conceitos mais importantes são discutidos no contexto de conjuntos funcionais. Os três conjuntos fundamentais são **primitivas gráficas, funções de segmentação e transformações window-viewport**.

Uma figura pode ser considerada como um conjunto de linhas e/ou cadeias de caracteres. As primitivas gráficas são um conjunto de funções que permitem ao usuário especificar em forma conveniente as linhas e caracteres que compõem a figura. Ainda que linhas e caracteres sejam os componentes básicos de uma figura, raramente são usados como primitivas na definição de uma figura. Um segmento é a primitiva adotada, sendo composto por um conjunto de linhas ou caracteres. Linhas ou caracteres num segmento podem ou não ser contíguos. Um segmento pode consistir de uma linha simples, um caráter simples, duas linhas separadas, e assim por diante. Uma figura usualmente tem um ou mais segmentos.

O conceito de segmento é usado para simplificar a criação e manipulação da figura. Contudo, é responsabilidade do usuário definir os tipos de segmentos para fazer um melhor uso dos mesmos em uma aplicação. Por exemplo, suponhamos que se queira construir uma figura que consista somente de triângulos e rectângulos e que triângulos e rectângulos sejam objetos independentes na figura. Assim, é natural manipular esses objetos individualmente. No nível conceitual, triângulos e rectângulos são os componentes básicos da figura, não linhas individuais. A capacidade de definição e criação de segmentos é provida pelas **funções de segmentação.**

Criar um segmento e torná-lo visível são duas operações diferentes. Tendo criado um segmento, este permanece invisível a menos que explicitamente se queira fazê-lo visível. Esta distinção permite ao programador de aplicação ter o controle sobre a figura conforme a necessidade do sistema gráfico. Um bom exemplo no qual esta capacidade é útil é na seleção de menu. Usualmente num sistema gráfico, o usuário tem muitos comandos para selecionar. Pode-se querer selecionar somente parte dos comandos ao mesmo tempo, neste caso, os comandos podem ser agrupados em um ou mais segmentos ou menus.

Em muitas aplicações, é necessário operar-se sobre as figuras, realizando escalamento, translação, rotação etc. Igualmente pode-se estar interessado em operações sobre partes da figura. Assim pode-se definir uma **window** na figura principal. A tela pode também ser dividida em diferentes partes. A window na figura principal pode ser mapeada, segundo as transformações mencionadas, em uma das partes da tela. A área da tela sobre a qual a **window** é mapeada é chamada uma **viewport**. O mapeamento desde uma window para uma viewport é chamado uma **transformação** window-viewport.

4.1.1 INTERFACE DO USUÁRIO

A interface do usuário é a parte do sistema ou programa de aplicação que determina como o usuário interage com o computador. Semelhantemente a todos os outros programas interativos, seus componentes são um fator determinante no sucesso ou fracasso do sistema proposto. Nos parágrafos que se seguem, algumas técnicas relevantes de interface do usuário para sistemas gráficos interativos são discutidas, avaliadas e uma escolha final é justificada.

Existem muitas formas de definir uma interface num sistema gráfico interativo. As mais comumente usadas são: a) teclado alfanumérico; b) funções de chave; c) técnicas de apontamento tais como o light-pen ou cursor; e d) uma combinação dos tipos listados. Descrição detalhada desses métodos pode ser encontrada em (/ NEWMAN 81/).

Se um teclado alfanumérico é usado como dispositivo de interface algum compilador ou interpretador é necessário para interpretar os comandos do usuário. Do ponto de vista do usuário, o uso de um teclado alfanumérico para a entrada de comandos tende a ser muito complicado e sujeito a erros. Isto porque: a) o usuário tem que lembrar toda a sintaxe de comandos do linguagem; b) uma grande quantidade de interações é requerida, tais como comandos de entrada, dados e assim por diante. Consequentemente, isto aumenta a possibilidade de erro do usuário.

Designar uma função específica para uma chave no teclado é uma idéia usada na técnica da função chave. Esta técnica é boa somente para sistemas gráficos muito simples. Também o número de funções requeridas pode ser maior que a quantidade de chaves disponíveis no teclado. A maior desvantagem é não prover uma interface fácil para o usuário. O usuário tem que lembrar todas as funções chaves disponíveis e os correspondentes operandos requeridos. Isto é particularmente indesejável desde que as operações esperadas sejam numerosas.

Uma das técnicas mais comuns para apontamento usada em um sistema gráfico interativo é chamada **seleção de menu**. Esta é uma técnica muito poderosa e natural para o controle interativo. Neste método, todos os comandos disponíveis para o usuário são agrupados em um conjunto de menus. Se mais de um menu está disponível, então num certo estágio, o menu apropriado é selecionado para display. Um menu é colocado sobre uma parte da tela enquanto o resto da tela é usado para mostrar outra informação. O usuário aponta para sua seleção com um dispositivo de entrada gráfico, o qual é usualmente um stylus, um cursor ou um light-pen.

Quase todos os sistemas gráficos interativos com algum grau de complexidade fazem uso de menu na sua seleção de operandos e comandos. Existem várias razões para isto. Primeiro, o menu permite exibir na tela o conjunto completo de opções válidas em um certo momento para o usuário. Segundo, permite uma maior flexibilidade. Diferentemente da técnica de função chave , um menu é facilmente mutável. Terceiro, reduz a interação do usuário ao mínimo Já que tanto para seleção de comandos e operações pode-se fazer a especificação por apontamento nos objetos apropriados. Isto resulta em uma menor quantidade de erros de entrada.

Na presente aplicação, para apontar para um comando específico, o sistema gera todos os objetos necessários tais como retângulos(conjunto entidades), losango(conjunto relacionamento) e demais. Os nomes de entidades, relacionamentos ou conjunto de valores não são previamente conhecidos.Isto requer que o usuário entre no sistema para especificar sua informação. Portanto, uma combinação de **teclado alfanumérico** e **seleção de menu** será utilizada na implementação.

Associado com esta aproximação está o problema de inconsistência na ação do usuário. A inconsistência surge porque o usuário algumas vezes requer entrada alfanumérica e às vezes requer posicionamento. Este problema pode ser resolvido projetando o sistema tal que a interação usuário-teclado seja bem definida e mínima.

4.1.2 FORMAS DE COMUNICAÇÃO HOMEM/MÁQUINA

Um sistema aceita entradas em uma variedade de formas, faz as transformações correspondentes produzindo as saídas também em diferentes formas. Entradas podem ser sinais de controle transmitidos por um tradutor, uma série de números fornecidos por um operador humano, um pacote de informação transmitido através de uma linha de comunicação ou um arquivo volumoso de dados retirados da memória secundaria. As transformações podem compreender desde uma simples comparação lógica até um algorítmo numérico complexo. Saídas podem ser através da representação gráfica ou texto no monitor ou páginas de informação.

As etapas de entrada-transformação-saída num sistema de computação, sem considerar o tamanho e complexidade dele podem ser melhor representados através de um diagrama de fluxo de dados. Sendo assim, utilizar-se-á esta metodologia para a definição do diálogo do sistema MER/PAC.

4.1.3 METODOLOGIA PARA ESPECIFICAÇÃO DO FLUXO DO DIÁLOGO

Na notação por fluxo de dados (/PRESSMAN 82/), a função total do sistema é representada como uma simples transformação de informação, representada na forma de um "nó". Assim, uma ou mais entradas, representadas por linhas rotuladas dirigem a informação a ser transformada para produzir informação de saída. Também pode ser notado que o modelo pode ser aplicado a um sistema inteiro ou somente a um elemento do software como é mostrado na figura 4.1.1.

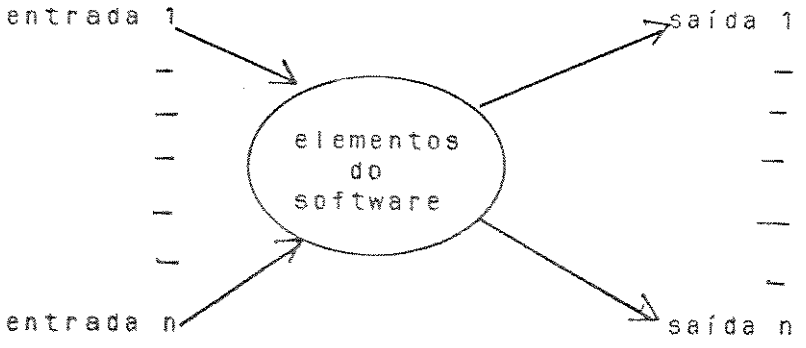


Figura 4.1.1 Transformação da informação

Onde:

. fluxo de dados, é representada por linhas rotuladas;

. processos (transformações) são representados por nós apropriadamente rotulados;

. fonte e destino de informação serão representadas como retângulos rotulados;

. armazenador de informação (ex.arquivo de dados) será representado por uma linha dupla horizontal;

. uma fonte de informação é uma localização de onde se originam dados (ex. um homem, um tradutor ou uma máquina);

. destino da informação é o destino final do dado que passa pelo sistema.

Pode-se concluir que num **diagrama de fluxo de dados** tem-se três atributos a considerar:

a. fluxo de informação num sistema (representado em forma manual, automática ou híbrida);

b. cada nó pode requerer maiores refinamentos para estabelecer um completo entendimento;

c. fluxo de dados mais que fluxo de controle é enfatizado.

Na definição de um sistema um grafo direcionado pode ser utilizado para representar como o conjunto de dados de entrada flui pelo sistema, transitando entre pontos de armazenamento e pontos de transformação, até a produção final do conjunto de dados de saída. Um exemplo de um tal grafo é (fig.4.1.2):

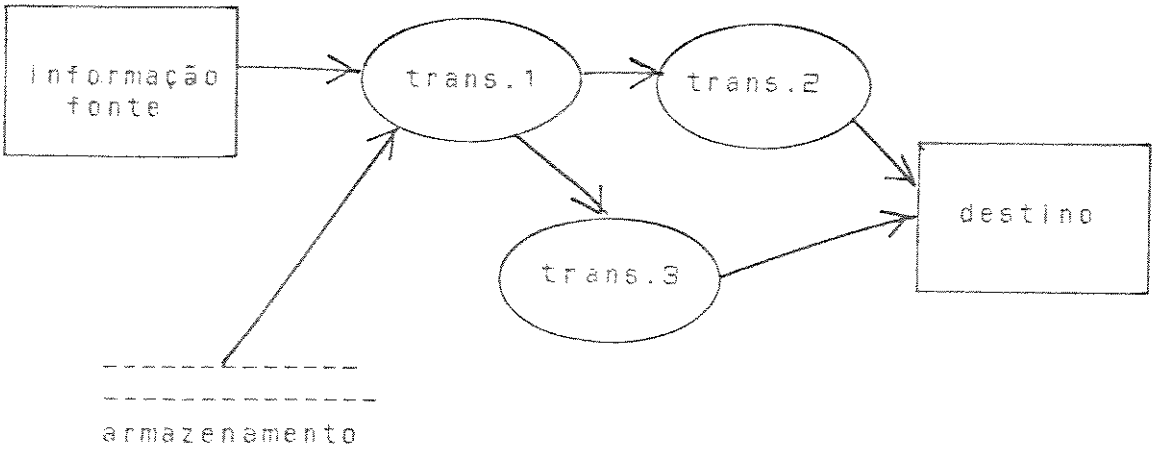


Figura 4.1.2 Grafo do fluxo de dados

Utilizando esta metodologia será definido a seguir o fluxo correspondente ao processo de definição e manipulação de dados do MER/PAC realizado através da LIGG e que é o tema deste trabalho.

4.2 ESPECIFICAÇÃO FUNCIONAL DA LIGG

4.2.1 INTRODUÇÃO

Definidas as convenções básicas para a representação de um diagrama de fluxo de dados, o sistema de diálogo, que implementará a LIGG, poderá numa visão geral ser definido pelo seguinte diagrama de fluxos (fig. 4.2.1).

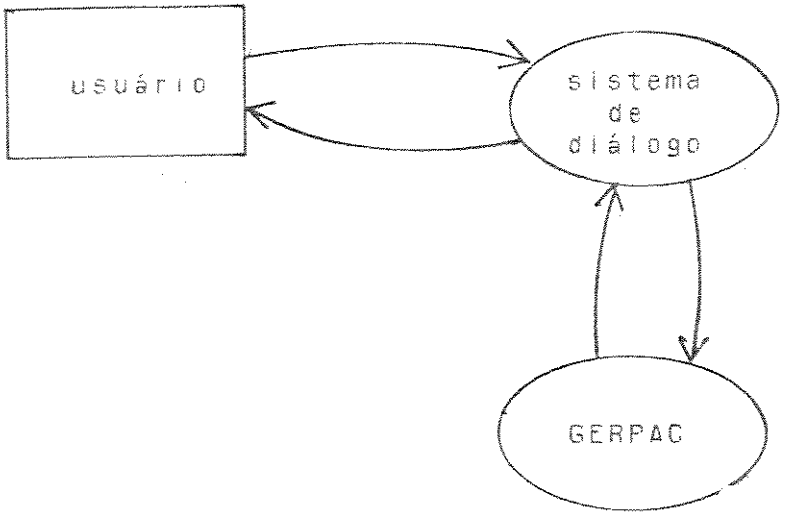


Figura 4.2.1 Grafo de fluxo da LIGG

Onde:

usuário: proporciona ao sistema a informação para iniciar o processo de diálogo.

sistema de diálogo: representa a interface de comunicação usuário-gerpac;

gerpac: faz o gerenciamento da informação a ser entregue à Base de Dados.

Considerando que a comunicação do usuário com o sistema será na forma interativa através de MENUS, o fluxo de dados num nível de maior detalhe pode ser representado da forma:(fig. 4.2.2)

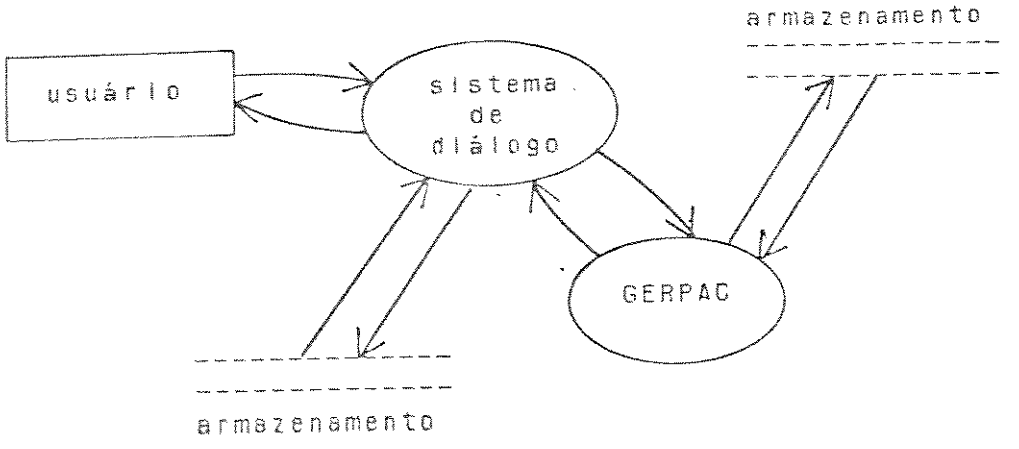


Figura 4.2.2 Fluxo de dados de LIGG

Um refinamento do sistema de diálogo permite mostrar claramente que ele além da interação com o usuário e o desenho das primitivas desejadas também se comunica com o sistema GERPAC para o processo de consulta e atualização da base de dados.(fig. 4.2.3)

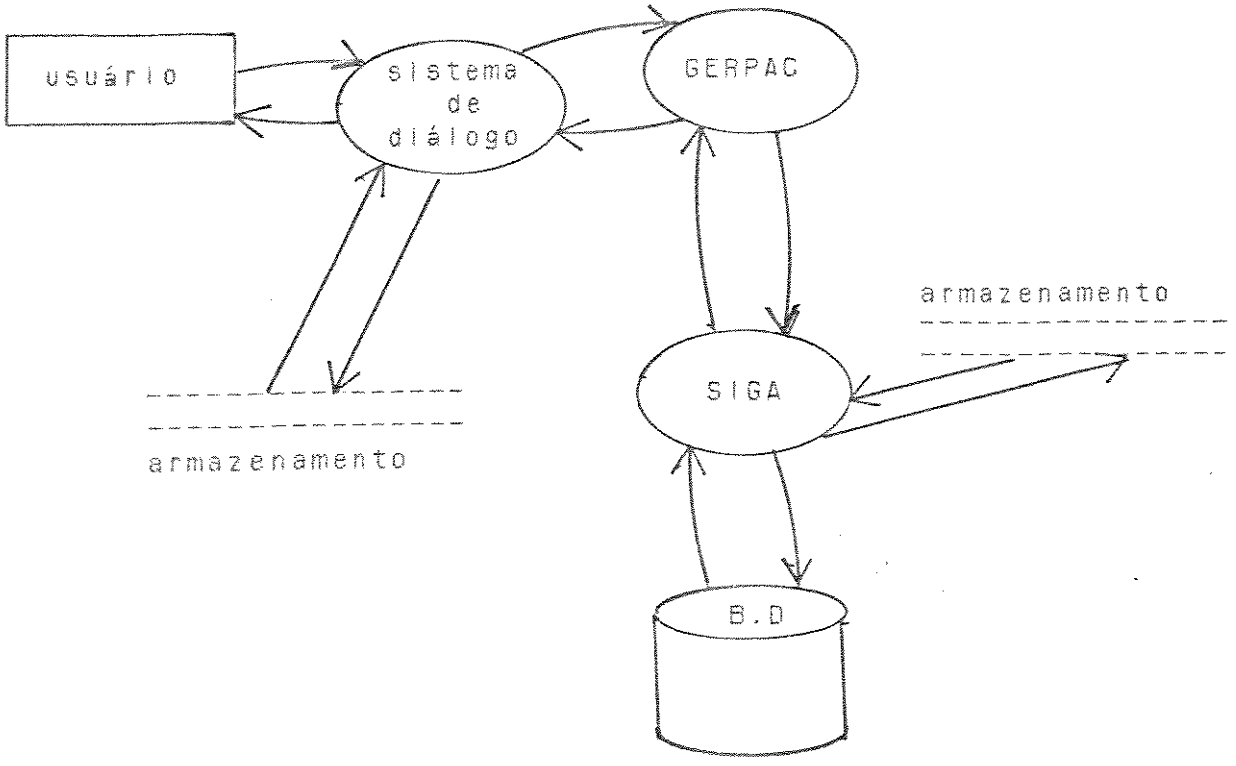


Figura 4.2.3 Fluxo de dados LIGG-BD

A figura 4.2.4 mostra em contornos gerais a operação do sistema GERPAC sendo apoiado pela LIGG nas fases de criação, manipulação e consulta da(s) base(s) de dados.

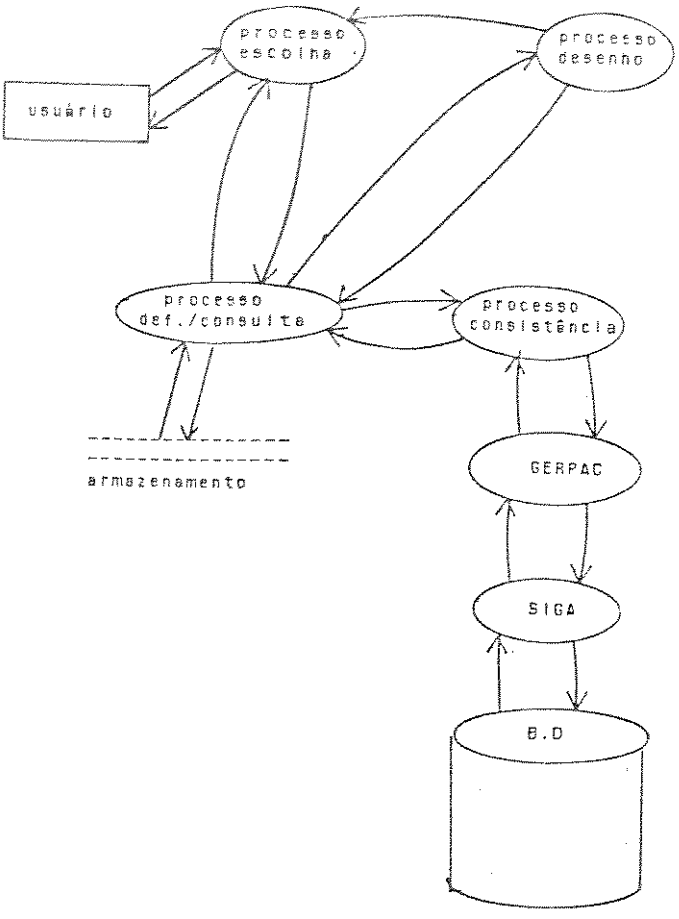


Figura 4.2.9 Sistema LIGG - GERPAC - BD

No grafo representativo do sistema pode-se observar os três principais estágios do fluxo da informação:

usuário: o que proporciona a informação fonte;

sistema de diálogo: onde se realizam as funções de escolha para o modo de operação (esquema/atributo), tipos de funções a realizar, desenho das primitivas e consistência da informação que será passada a GERPAC;

gerpac-siga-banco de dados: fazendo o gerenciamento da informação (GERPAC), especificando o como fazê-la (SIGA) e seu posterior armazenamento (B.D).

Assim o sistema de diálogo que oferece a LIGG como instrumento de comunicação ao usuário, após os procedimentos de consistência comunica-se com o GERPAC que através do SIGA acessa a base de dados.

A seção seguinte descreve em detalhes a LIGG e o sistema de diálogo.

4.2.2 ESPECIFICAÇÃO DO SISTEMA DE DIÁLOGO

Como foi dito anteriormente, a LIGG é uma interface gráfica que permite a definição e manipulação dos elementos de um banco de dados MER/PAC. Sendo definida sob este contexto, ela deverá apresentar ao usuário uma forma amigável de comunicação com o sistema.

A forma de comunicação homem/máquina oferecida será interativa utilizando menus e apontamento nas primitivas a definir ou manipular.

A título de ilustração será apresentada um protótipo de implementação parcial da LIGG onde se utilizará o ambiente GKS (Graphical Kernel System) sobre o sistema GAV (/HARTELT 86/).

Inicialmente, mostrar-se-á o ambiente gráfico que a LIGG oferecerá ao usuário.

Ao iniciar o sistema o usuário recebe o seguinte menu (fig. 4.2.2.1).

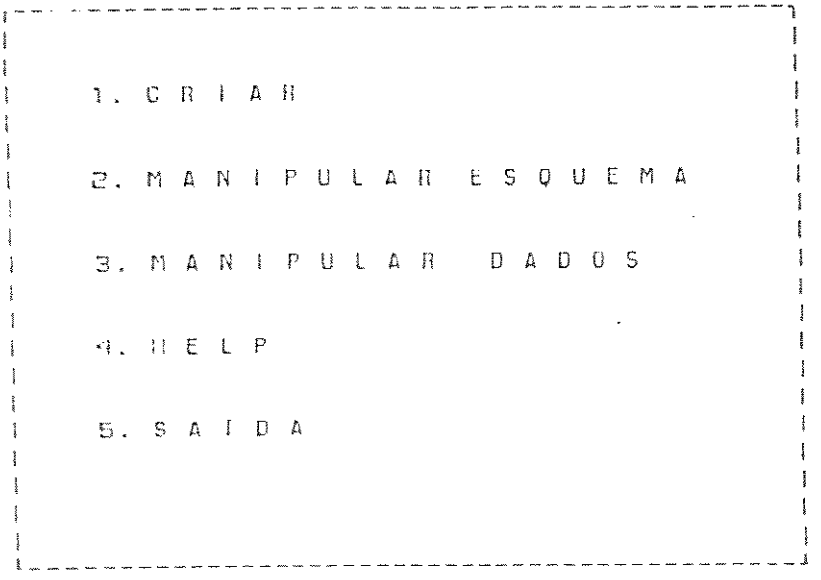


Figura 4.2.2.1 Menu inicial

Se a opção escolhida for a quinta, o usuário deseja sair do sistema e o programa conclui sua execução. Se a opção escolhida for a primeira, segunda ou terceira o programa solicita do usuário as seguentes informações.(fig.4.2.2.2)

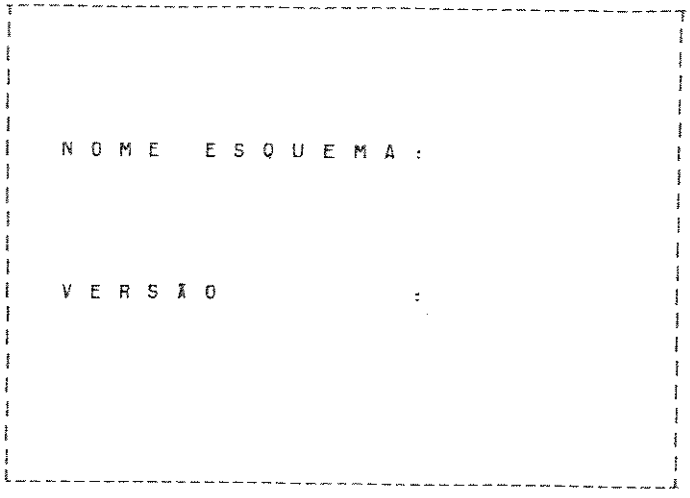


Figura 4.2.2.2 Definição esquema/versão

A segunda e a terceira opções permitem a manipulação de esquemas e dados de uma base de dados anteriormente definida. A quarta opção fornece informações sobre a operação do sistema (fig. 4.2.2.3).

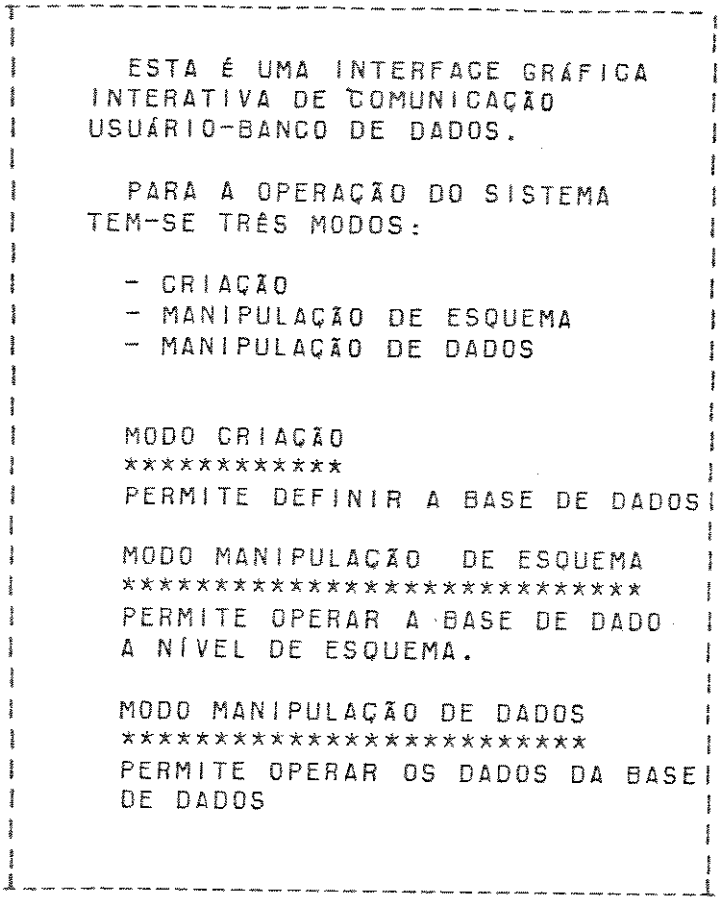


Figura 4.2.2.3 Help

Se a opção escolhida for a primeira, após o preenchimento do nome do esquema e da versão, o sistema apresenta ao usuário o seguinte menu.(fig.4.2.2.4)

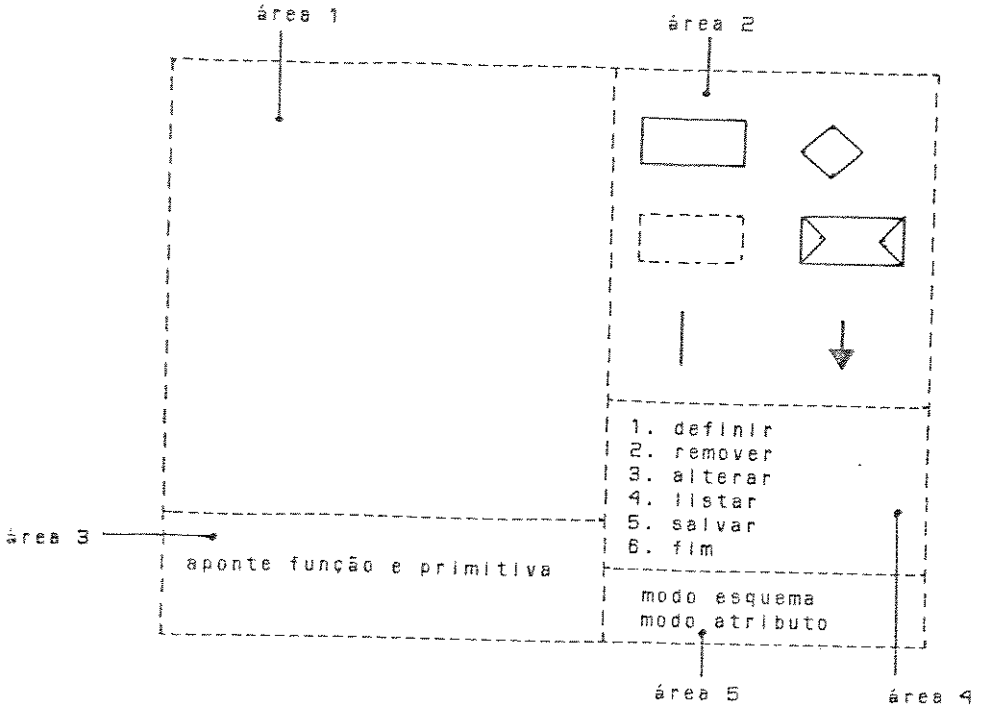


Figura 4.2.2.4 Criação esquema/atributo

Onde, a área 1 é a área de trabalho que será utilizada pelo usuário para o desenho de esquemas e manipulação dos dados. A área 2 apresenta o menu dos tipos de primitivas disponíveis (entidades, relacionamentos, esquema, agrupamento). A área 3 solicita ao usuário que ele escolha a função e a primitiva a ser manipulada. A área 4 apresenta as funções que o usuário pode realizar na função de criação. A área 5 apresenta os modos de operação sobre esquema ou atributo:

Se o modo de criação é referido ao modo esquema, os menus das figuras 4.2.2.4, 4.2.2.5 e 4.2.2.6 são apresentados ciclicamente durante o processo de criação, podendo o usuário executar as funções definidas na área 4. O ciclo finaliza quando o usuário escolher a opção 6.

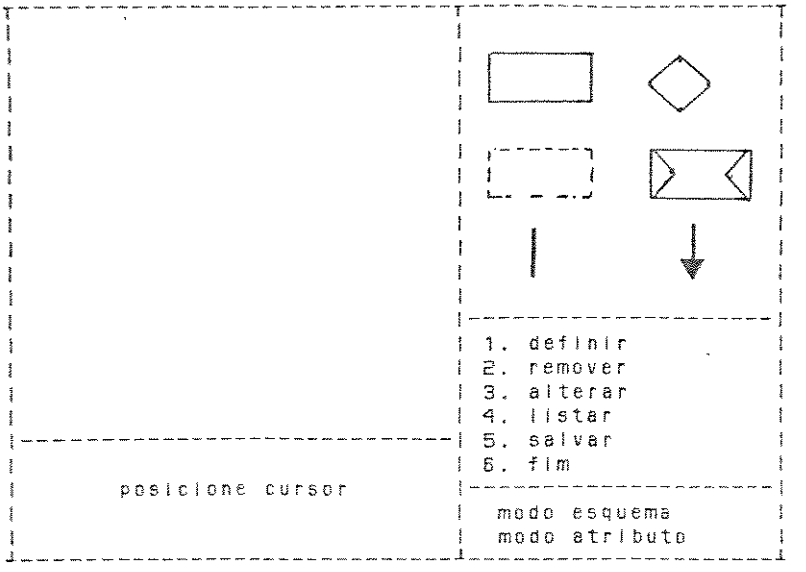


Figura 4.2.2.5 Posicionamento cursor

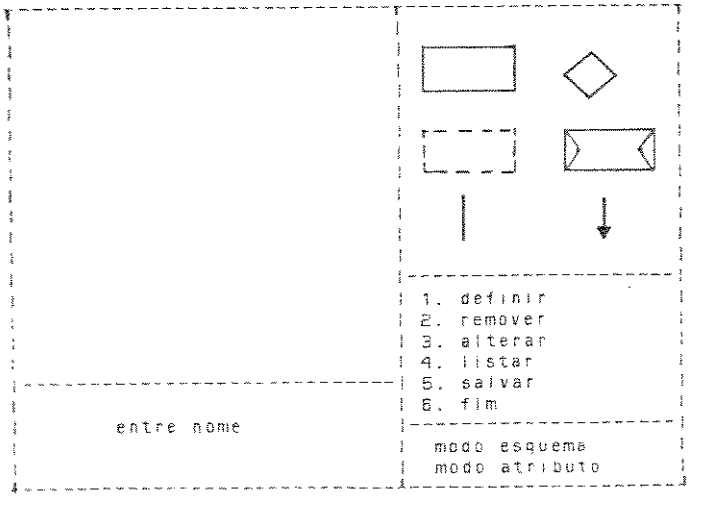


Figura 4.2.2.6 Solicitação nome

Se o modo de criação escolhido é referido a atributos, o usuário terá na tela os menus das figuras 4.2.2.7 e 4.2.2.8, onde na área 1 será mostrado o esquema requerido. A consulta a cada elemento do esquema é feita por apontamento. O processo termina ao se escolher a opção 6.

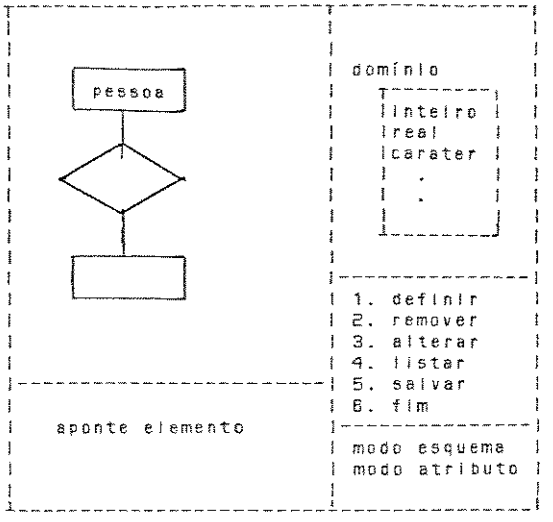


Figura 4.2.2.7 Apontamento primitiva

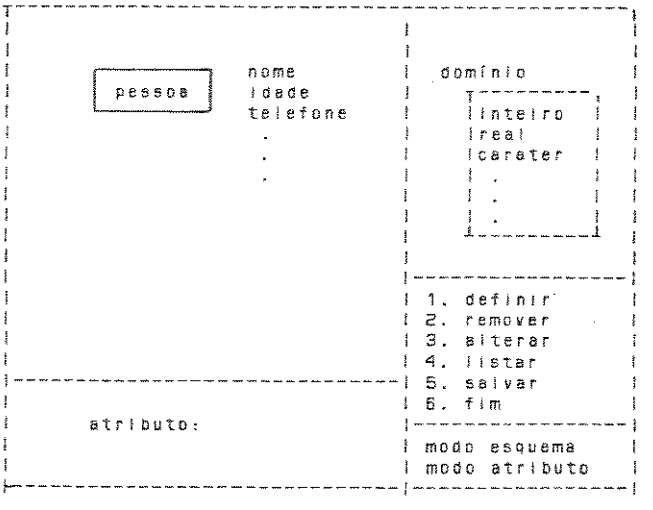


Figura 4.2.2.8 Indicação atributo

Se a opção escolhida no menu inicial for a segunda, o sistema depois de passar pelo menu da figura 4.2.2.2 apresenta o menu da figura 4.2.2.4, só que na área 1 já existe o esquema definido anteriormente (na criação). A partir desse ponto o usuário pode realizar a operação de manipulação no modo esquema, gerando as telas das figuras 4.2.2.5 e 4.2.2.6 ou a manipulação no modo atributo gerando as telas das figuras 4.2.2.7 e 4.2.2.8 de forma análoga à criação. O processo de manipulação termina ao se escolher a opção 6.

A figura 4.2.2.9 é gerada quando da escolha da opção três(manipulação de dados).

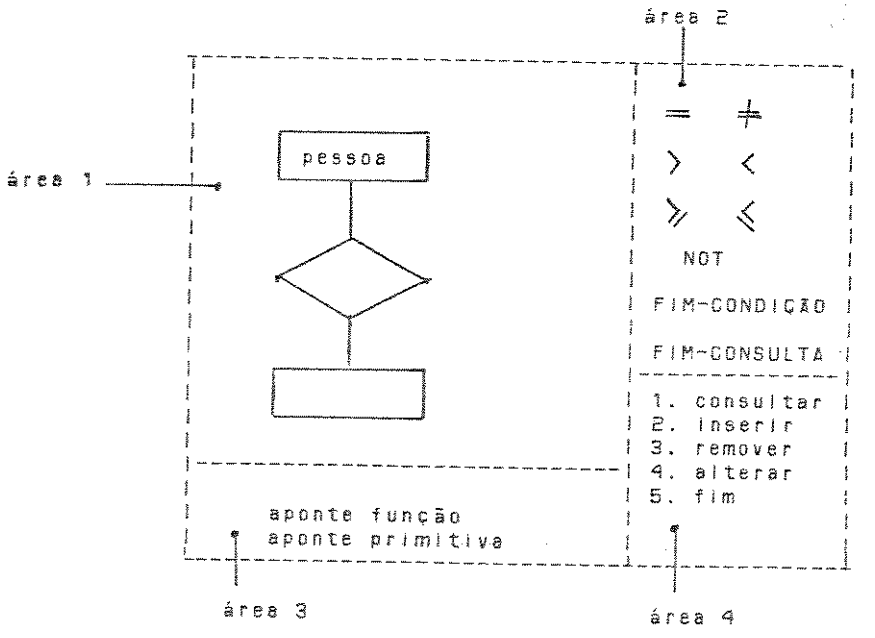


Figura 4.2.2.9 Manipulação de dados- menu inicial

Onde na área 1 é mostrado o esquema sendo consultado. A área 2 apresenta os operadores existentes. A área 3 solicita ao usuário que ele escolha as funções e primitivas a manipular. A área 4 apresenta as funções que o usuário pode executar no processo de manipulação dos dados.

Uma vez escolhida a função e primitiva o sistema gera o seguinte menu.(fig. 4.2.2.10).

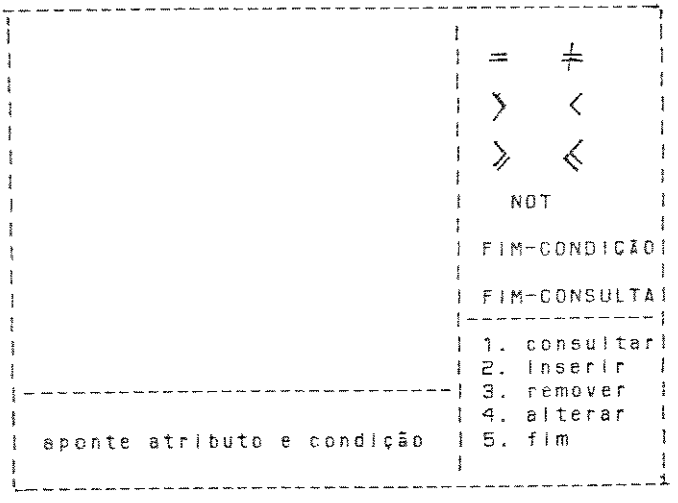


Figura 4.2.2.10 Especificação atributos

Onde o usuário através das funções da área 4 faz a manipulação dos dados desejados. O processo finaliza se o usuário escolher a opção 5, com o qual o sistema volta ao menu inicial.

A operação do sistema de diálogo será apoiada em uma estrutura de dados conforme exposto a seguir.

4.2.3 ESTRUTURA DE DADOS DA LIGG

Para poder oferecer aos usuários todos os recursos mencionados, a LIGG deve dispor de uma estrutura de dados, onde devem estar armazenadas todas as informações necessárias a sua operação.

Esta estrutura de dados é composta por cinco tabelas, cujo conteúdo se especifica na continuação.

A tabela 4.2.1 contém informações sobre as entidades definidas. Nela são armazenados o nome da entidade, seu tipo e atributos.

Na tabela 4.2.2 encontram-se as informações referentes aos relacionamentos definidos, tais como nome, tipo e atributos.

A tabela 4.2.3 contém informações sobre as associações estabelecidas entre as primitivas. Nela são armazenados o nome da associação, o nome e classe da entidade envolvida, o nome do relacionamento participante na associação, a cardinalidade e o tipo de associação definida.

Na tabela 4.2.4 encontram-se as informações referentes aos agrupamentos. Nela estão contidos o nome do agrupamento, seu tipo, o nome e classe dos elementos componentes do agrupamento e as regras que os definem.

A tabela 4.2.5 contém informações sobre os esquemas definidos. Nela são armazenados o nome do esquema, o tipo e atributos ou versões.

nome	tipo	atributo

Tabela 4.2.1 Entidades

nome	tipo	atributo

Tabela 4.2.2 Relacionamentos

nome	entidade .nome	entidade .classe	nome-rel	cardinalidade	a.tipo

Tabela 4.2.3 Associações

nome	tipo	elemento nome	elemento classe	regra

Tabela 4.2.4 Agrupamentos

esq.nome	esq.tipo	esq.atributo

Tabela 4.2.5 Esquemas

Para o processo de manipulação a LIGG também dispõe de um conjunto de tabelas.

Esta estrutura de dados é composta por quatorze tabelas divididas em 3 grupos correspondentes a:

a. Tabelas para modificação de elementos existentes (modo esquema), cujo conteúdo se especifica a seguir.

Tabela 4.2.6 contendo informações sobre os elementos acrescentados. Nela são armazenados o nome e a classe dos elementos acrescentados.

Tabela 4.2.7 contém informações sobre os elementos acrescentados num agrupamento. Estão contidos nela o nome do agrupamento, o nome e classe do elemento acrescentado ao esquema.

A tabela 4.2.8 contém informações sobre os elementos removidos. Nela são armazenados o nome e a classe dos elementos removidos.

Na tabela 4.2.9 encontram-se informações sobre os elementos removidos de um agrupamento. Estão contidos nela o nome do agrupamento e o nome e classe do elemento removido.

A tabela 4.2.10 contém informações sobre os elementos alterados. Nela são armazenados o antigo e o novo nome.

b. Tabelas para modificação dos atributos/domínios. O conteúdo destas tabelas é:

A tabela 4.2.11 contém informações sobre os atributos acrescentados. Nela são armazenados o nome do atributo acrescentado ao esquema e o domínio correspondente.

Na tabela 4.2.12 as informações contidas são referidas aos atributos removidos. Ela contém o nome do atributo e os domínios removidos.

A tabela 4.2.13 contêm informações sobre os nomes dos atributos alterados.

Na tabela 4.2.14 encontram-se as informações sobre os domínios alterados. Nela é armazenado o tipo do domínio alterado.

c. Tabelas para remoção das primitivas:

As tabelas 4.2.15, 4.2.16 e 4.2.17 contêm os nomes das entidades,associações e relacionamentos a remover.

A tabela 4.2.18 contém todos os nomes das associações que se relacionam com a entidade que define a associação.

A tabela 4.2.19 contém todos os nomes das associações que se relacionam com o relacionamento que define a associação.

nome	classe

Tabela 4.2.6 Elementos acrescentados

	elemento	
nome	nome	classe

Tabela 4.2.7 Elementos acresc. a agrupamento

nome	classe

Tabela 4.2.8 Elementos removidos

	elemento	
nome	nome	classe

Tabela 4.2.9 Elementos remov. do agrupamento

nome antigo	nome novo

Tabela 4.2.10 Elementos alterados

atrib.nome	atrib.dominio

Tabela 4.2.11 Atributos acrescentados

atrib.nome	atrib.domínio

Tabela 4.2.12 atributos removidos

atrib.nome

Tabela 4.2.13 Atrib. alterados

domínio.tipo

Tabela 4.2.14 Dom. alt.

nome

Tabela 4.2.15 Ent. a remover

nome

Tabela 4.2.16 Assoc. a remover

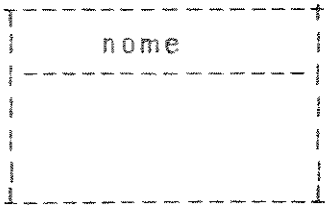


Tabela 4.2.17 Relac. a remover

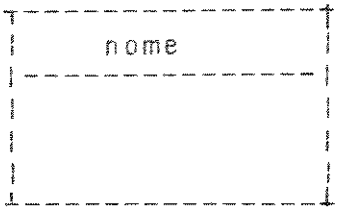


Tabela 4.2.18 Associações

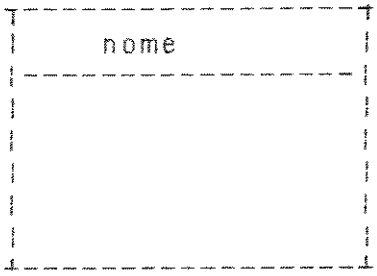


Tabela 4.2.19. Associações

4.2.4 PROJETO DE IMPLEMENTAÇÃO DA LIGG

Neste ítem, a LIGG é especificada a nível funcional. Nesta especificação constam para cada procedimento:

- nome;
- objetivo;
- parâmetros de entrada;
- parâmetros de saída;
- erros

Para uma melhor visão geral, apresenta-se o diagrama de hierarquia dos procedimentos definidos. Assim as figuras X-A, X-B, X-C apresentam a árvore de chamadas, com os nomes das rotinas codificadas (A, B, C,...). Em seguida a relação dos nomes dos procedimentos é apresentada.

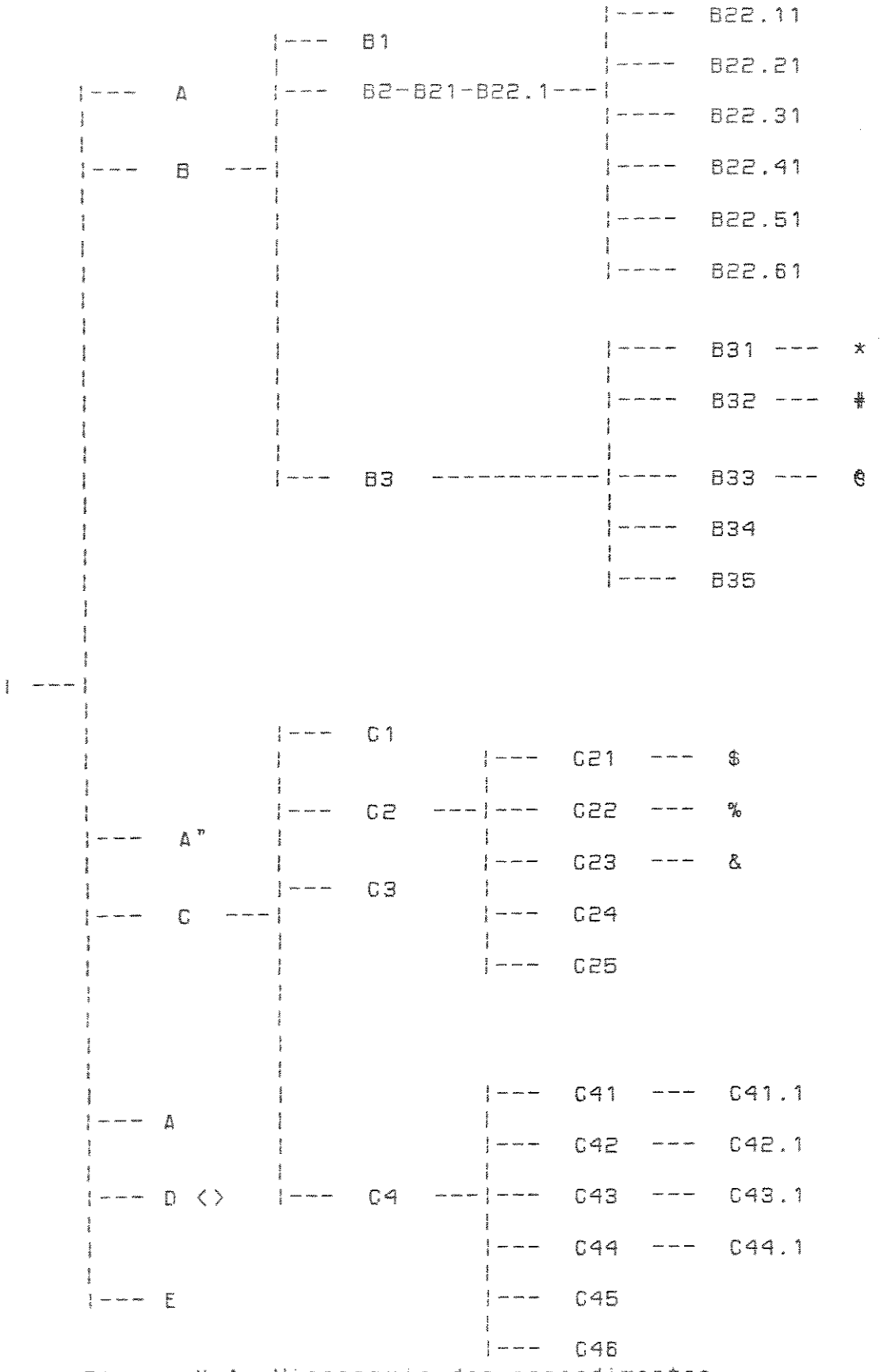


Figura X-A: Hierarquia dos procedimentos

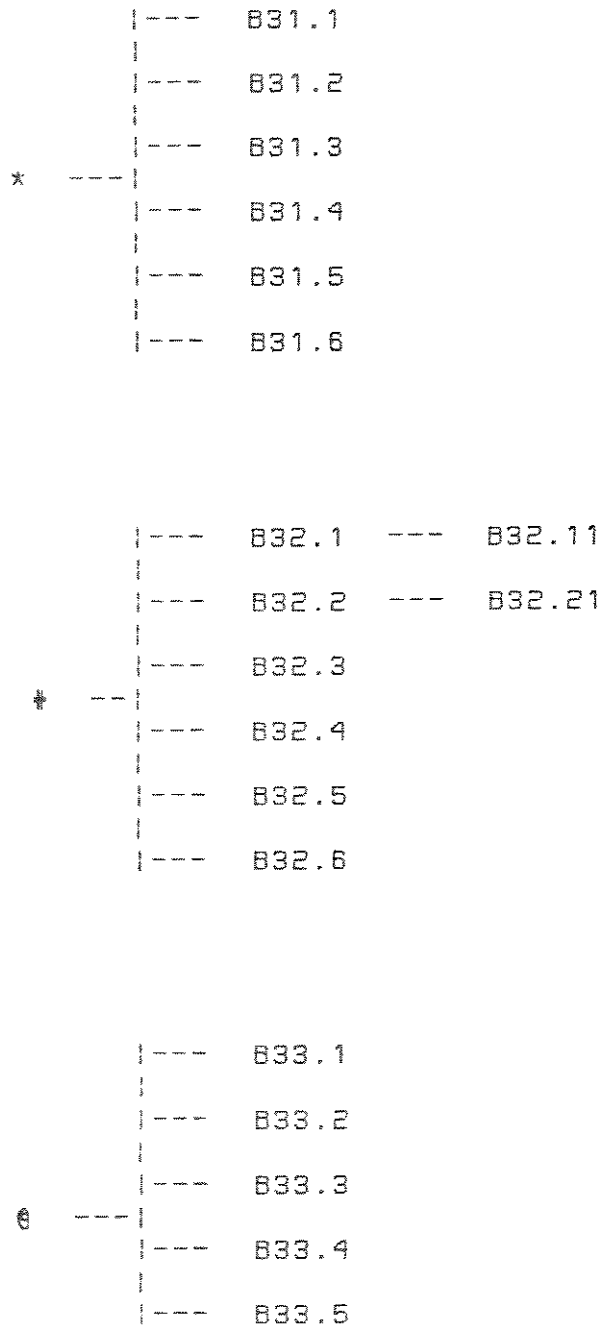


Figura X-8: Hierarquia dos procedimentos (continuação)

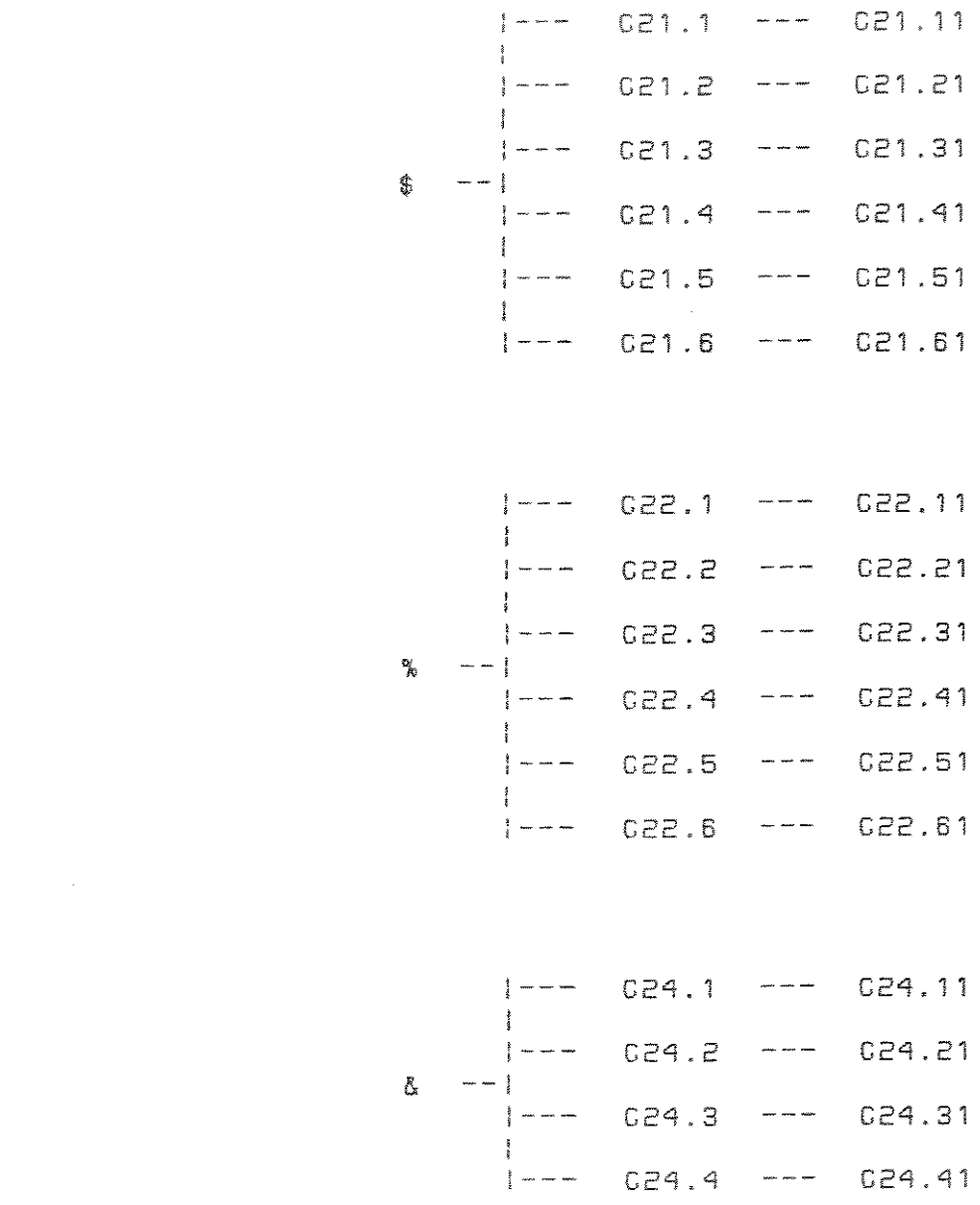


Figura X-C: Hierarquia dos procedimentos (continuação)

RELAÇÃO DOS NOMES DOS PROCEDIMENTOS

I: INICIAL

A: PROC-ESCREVE-TELA-INICIAL

B: PROC-CRIAR

B1: PROC-TELA-CRIAR

B2: PROC-CRIA-MODO-ATRIBUTO

B3: PROC-OPERA-MODO-ESQUEMA

B21: PROC-APONTA-ELEMENTO

B22.1: PROC-OPERA-MODO-ATRIBUTO

B22.11: PROC-DEFINE-ATRIBUTO

B22.21: PROC-REMOVE-ATRIBUTO

B22.31: PROC-ALTERA-ATRIBUTO

B22.41: PROC-ALTERA-DOMÍNIO

B22.51: PROCEDIMENTO-LISTAR

B22.61: PROCEDIMENTO-SALVAR

B31: PROCEDIMENTO-DEFINIÇÃO-ESQUEMA

B32: PROCEDIMENTO-REMOÇÃO-ESQUEMA

B33: PROCEDIMENTO-ALTERAÇÃO-ESQUEMA

B34: PROCEDIMENTO-LISTAR

B35: PROCEDIMENTO-SALVAR

*

B31.1: DEFINE-ENTIDADE

B31.2: DEFINE-RELACIONAMENTO

B31.3: DEFINE-ASSOCIAÇÃO

B31.4: DEFINE-AGRUPAMENTO

B31.5: DEFINE-ELEMENTO-AGRUPAMENTO

B31.6: DEFINE-ESQUEMA

#

B32.1: REMOVE-ENTIDADE

B32.2: REMOVE-RELACIONAMENTO

B32.3: REMOVE-ASSOCIAÇÃO

B32.4: REMOVE-AGRUPAMENTO

B32.5: REMOVE-ELEMENTO-AGRUPAMENTO

B32.6: REMOVE-ESQUEMA

@

B33.1: ALTERA-ENTIDADE

B33.2: ALTERA-RELACIONAMENTO

B33.3: ALTERA-ASSOCIAÇÃO

B33.4: ALTERA-AGRUPAMENTO

B33.5: ALTERA-ESQUEMA

B32.11: TESTE-REMOÇÃO-ENTIDADE

B32.21: TESTE-REMOÇÃO-RELACIONAMENTO

A": PROC-TELA-NOME-ESQUEMA

C: PROC-MANIPULAR-ESQUEMA

C1: PROC-TELA-CRIAR

C2: MODO-ESQUEMA-M

C3: DESENHA-TELA-ATRIBUTO

C4: MODO-ATRIBUTO-M

C21: PROCEDIMENTO-ACRESCENTAR-ESQ-M

C22: PROCEDIMENTO-REMOVER-ESQ-M

C23: PROCEDIMENTO-ALTERAR-ESQ-M

C24: PROCEDIMENTO-LISTAR

C25: PROCEDIMENTO-SALVAR

C41:PROC-ACRESCENTA-ATRIBUTI-ESQ-EXI

C42: PROC-REMOVE-ATRIBUTO-ESQ-EXI

C43: PROC-ALTERA-ATRIBUTO-ESQ-EXI

C44: PROC-ALTERA-DOMÍNIO-ESQ-EXI

C45: PROCEDIMENTO-LISTAR

C46: PROCEDIMENTO-SALVAR

$

C21.1: PROC-ACRESCENTAR-ENTIDADE

C21.2: PROC-ACRESCENTAR-RELACIONAMENTO

C21.3: PROC-ACRESCENTAR-ASSOCIAÇÃO

C21.4: PROC-ACRESCENTAR-AGRUPAMENTO

C21.5: PROC-ACRESCENTAR-ELEMENTO AGRUPA-
MENTO

C21.6: PROC-ACRESCENTAR-ESQUEMA

%

C22.1: PROC-REMOVER-ENTIDADE

C22.2: PROC-REMOVER-RELACIONAMENTO

C22.3: PROC-REMOVER-ASSOCIAÇÃO

C22.4: PROC-REMOVER-AGRUPAMENTO

C22.5: PROC-REMOVER-ELEMENMTO-AGRUPA-
MENTO

C22.6: PROC-REMOVER-ESQUEMA

&

C24.1: PROC-ALTERAR-ENTIDADE

C24.2: PROC-ALTERAR-RELACIONAMENTO

C24.3: PROC-ALTERAR-ASSOCIAÇÃO

C24.4: PROC-ALTERAR-AGRUPAMENTO

C41.1: PROC-DEFINE-ATRIBUTO

C42.1: PROC-REMOVE-ATRIBUTO

C43.1: PROC-ALTERA-ATRIBUTO

C44.1: PROC-ALTERA-DOMÍNIO

C21.11: DEFINE-ENTIDADE

C21.21: DEFINE-RELACIONAMENTO

C21.31: DEFINE-ASSOCIAÇÃO

C21.41: DEFINE-AGRUPAMENTO

C21.51: DEFINE-ELEMENTO-AGRUPAMENTO

C21.61: DEFINE-ESQUEMA

C22.11: REMOVE-ENTIDADE

C22.21: REMOVE-RELACIONAMENTO

C22.31: REMOVE-ASSOCIAÇÃO

C22.41: REMOVE-ENTIDADE

C22.51: REMOVE-ELEMENTO-AGRUPAMENTO

C22.61: REMOVE-ENTIDADE

C24.11: ALTERA-ENTIDADE

C24.21: ALTERA-RELACIONAMENTO

C24.31: ALTERA-ASSOCIAÇÃO

C24.41: ALTERA-AGRUPAMENTO

D: PROC-MANIPULA-DADOS

D1: PROC-TELA-MANIPULA-DADOS

D2: PROC-LER-ESQUEMA

D3: PROC-CONSULTAR

D4: PROC-INSERIR

D5: PROC-ALTERAR

D6: PROC-REMOVER

E: PROC-HELP

No anexo B apresentam-se as especificações e no anexo C o detalhamento dos procedimentos da LIGG. Para isso, será utilizada uma linguagem para especificação de projetos que possui estruturas de controle semelhantes às oferecidas nas linguagens de alto nível.

5 PROTÓTIPO DE IMPLEMENTAÇÃO DA LIGG

Como forma de ilustrar uma possível implementação da LIGG através de um sistema de diálogo, este capítulo define uma implementação parcial, realizada utilizando-se o ambiente GAV, Gerenciador de Áreas de Visualização, instalado sobre um sistema GKS.

O próximo ítem fornece uma descrição passo a passo deste sistema, assim como o resultado de sua implementação.

5.1 DESCRIÇÃO DO SUB-CONJUNTO IMPLEMENTADO

PASSO 1: Neste passo o menu de entrada (fig.4.2.2.1) é apresentado ao usuário. Se a opção escolhida for a quinta, o usuário deseja sair do sistema e o programa conclui sua execução. Se a opção escolhida for a primeira, segunda ou terceira, o passo dois será executado. A segunda e terceira opções permitem a manipulação de esquemas e dados de uma base de dados anteriormente definida. A quarta opção fornece informações sobre a operação do sistema.

PASSO 2: Após a escolha das opções 1 (tema deste exemplo), 2 ou 3, o programa solicita do usuário as seguintes informações (fig.4.2.2.2).

- nome do esquema
- versão

PASSO 3: A seguir a tela da figura 4.2.2.4 é apresentada ao usuário. A área 1 é a área de trabalho que será utilizada pelo usuário para o desenho de esquemas e manipulação da base de dados. A área 2 apresenta o menu de tipos de primitivas disponíveis (entidades, relacionamentos, esquema, agrupamento). A área 3 solicita ao usuário que ele escolha a função e primitiva a ser manipulada. A área 4 apresenta as funções que o usuário pode realizar na função de criação. A área 5 apresenta os modos de operação do sistema. Se a opção 6 for escolhida o passo 1 é o próximo passo a ser executado e o menu de entrada é novamente apresentado. Caso contrário, ou seja, a escolha seja a opção 1 o passo 4 é realizado.

PASSO 4: Neste passo o programa solicita ao usuário que posicione o cursor na posição desejada a fazer o desenho, (fig. 4.2.2.5). A primitiva selecionada é desenhada a partir da posição fornecida, e o programa solicita ao usuário que ele entre o nome dela (fig.4.2.2.6).

Uma vez que o usuário entra o texto, o programa solicita novamente que se posicione o cursor no lugar onde se vai escrever o texto. Entrado o texto, o control volta ao passo 3 iniciando-se o ciclo de execução.

Se o usuário escolher a alternativa 6 da área 4, o passo 1 é o próximo passo a ser executado e o menu de entrada é novamente apresentado. Escolhendo a alternativa 5 do menu, o programa conclui a execução.

5.2 DESCRIÇÃO DA IMPLEMENTAÇÃO

Um primeiro passo na implementação do exemplo apresentado foi a definição da LIGG através de uma linguagem de especificação de projetos que possui estruturas de controle semelhantes às oferecidas numa linguagem de alto nível.

A implementação desses procedimentos foi feita através de procedimentos definidos em FORTRAN 77, utilizando-se o ambiente GAV (Gerenciador de Áreas de Visualização), instalado sobre um sistema GKS (Graphical Kernel System).

Para realizar as diversas funções próprias do projeto foi definida uma estrutura de dados, na qual é armazenada a informação a ser passada posteriormente ao GERPAC.

Os elementos de hardware utilizados foram um terminal gráfico VT-240 ligado ao Sistema VAX-785 da UNICAMP.

O anexo A apresenta a implementação realizada.

6. CONSIDERAÇÕES FINAIS

A linguagem gráfica baseada em grafos para o MER/PAC (LIGG) proposta neste trabalho de tese, teve sua origem no modelo de dados entidade-relacionamento proposto por CHEN em 1976. A partir de uma extensão deste modelo voltada para aplicações PAC (MER/PAC) proposto por Delgado (/DELGA 87/), se definiu a LIGG, tendo como um dos seus objetivos ser uma interface amigável na comunicação usuário-banco de dados.

O presente trabalho no decorrer de seus capítulos procurou dar ao leitor uma visão geral de temas afins que na literatura geralmente se apresentam como mundos separados e que na realidade para o usuário formam um só conjunto. Esses temas abordados no trabalho são os de projeto de base de dados - modelos de dados - linguagens e comunicação usuário/banco de dados. Um dos méritos do presente trabalho é precisamente este, no sentido de iniciar o estudo assinalando quais devem ser os critérios para a escolha de um modelo de dados, fazendo uma rápida análise das linguagens de manipulação de dados mais conhecidas implementadas, para finalmente fazer uma proposta diferente das existentes, na qual a especificação e comunicação homem/máquina seja a mais natural possível. Para isto o trabalho se utiliza de elementos da computação gráfica.

O trabalho no seu desenvolvimento foi dividido em quatro grandes tópicos:

1. Definição da LIGG
2. Especificação funcional da LIGG
3. Detalhamento dos procedimentos da LIGG
4. Implementação de um exemplo da LIGG.

Para apoiar a realização das diversas funções relacionadas ao projeto consulta/manipulação de banco de dados PAC, definiu-se uma estrutura de dados interna, na qual é armazenada a informação, que depois de uma análise de consistência é passada ao sistema gerenciador para PAC para seu posterior processamento.

No exemplo de implementação a LIGG faz o reconhecimento do texto e primitivas utilizando-se de procedimentos definidos em FORTRAN 77 e chamadas a rotinas gráficas do pacote gráfico GKS (Graphical Kernel System), sobre o ambiente GAV (Gerenciador de Áreas de Visualização).

Entre as contribuções do presente trabalho pode-se assinalar:

1. Definição de uma linguagem de definição/consulta/manipulação que utiliza como ferramenta de interação o próprio modelo que o usuário tem de sua aplicação.

2. Caracterização das diversas opções apresentadas durante a utilização da LIGG, em função das entradas possíveis em cada instante, obtendo-se assim uma minimização dos procedimentos de consistência.

3. Agregação de procedimentos de definição/manipulação dos objetos durante a vida do projeto, como exigido em PAC.

4. Apresentação da metodología de desenvolvimento da LIGG, com o fornecimento de sua especificação funcional e do projeto de implementação.

É importante frisar que a solução adotada para a definição desta interface gráfica sai do campo habitual de pesquisa nesta área, onde a maior parte das interfaces definidas até agora são de tipo procedural. Embora este ponto tenha causado alguma dificuldade no desenvolvimento do trabalho, pela falta de bibliografía correspondente a implementações de interfaces não procedurais para PAC/BD, crê-se que interfaces não-procedurais constituirão importante campo de trabalho relacionado a linguagens de banco de dados.

Ficam como pontos abertos para próximos trabalhos a implementação completa do sistema gráfico LIGG, como também a possibilidade dessa implementação ser feita utilizando ferramentas de software e hardware mais versáteis como por exemplo a linguagem de programação C e dispositivos de entrada tais como o mouse ou light-pen, que em seu conjunto permitirão ter um sistema mais dinâmico na comunicação usuário-banco de dados.

Outro ponto em aberto e não abordado neste trabalho, é o tratamento sob o ponto de vista de esquema, correlacionando-se os diferentes níveis da hierarquia de un projeto.

7. BIBLIOGRAFIA

/BERZ 83/ : BERZTISS, A. T. and THATTE, S."Specifications and Implementation of Abstract Data Types".Advances in Computers,vol.22, pp.295-353, Academic Press, 1983.

/BOYCE 74/ : BOYCE, R.F. and CHAMBERLIN, D.D King III and M.M HAMMER,"Specifying queries as relational expressions": SQUARE. In Database Management Proc.IFIP Working Conf., April 1974, International Federation for Information Processing, Geneva,1974, pp.169-177.

/CHALLIS 82/ : CHALLIS, M. F." Typing in Data Base Models" In /ENCA 82/, pp. 265-279.

/CHAMBERLIN 74/: CHAMBERLIN, D.D. and BOYCE, R.F " SEQUEL: A structured English query language, Proc".ACM SIGFIDET Workshop Ann Arbor, MI May 1974, Special Interest Group on File Description and Translation, Association for Computing Machinery, 1974.

/CHAM 76/ : CHAMBERLIN, D. D. Relational Data Base Management Systems, Computing Surveys,Vol.8, No1, March 1976.

/CHAMBERLIN 75/: Chamberlin, D. D."Implementation of a Structured English Query Language" Comm. ACM Vol.18, No 10,october 1975.

/CHEN 76/ : CHEN,P.P. S. "The Entity-Relationship Model Toward a Unified View of Data," ACM Trans. on Database Systems, vol.1, No1, 1976.

/CHEN 77/ : CHEN, P. P. S. "The Entity-Relationship Approach to logical Data Base Design", The Q.E.D. Monograph Series-Data Base Management No6, 1977.

/CODASYL 78/ : Report of the CODASYL Data Description Language Committee, Inf. Syst., 3(4) 1978 pp.247-320.

/CODD 70/ : CODD, E. F. A Relational Model of Data for Large Shared data Banks. Com. ACM Vol.13, 6 June 1970.

/CODD 71/ : CODD, E.F. " Relational Completeness of Data Base Sublanguages"- Courant Computer Science Symposium, May 1971.

/CRESTIN 79/ : CRESTIN, J.P. and LUCAS, M. " What Might be Computer Graphics",ed. Guldy, R.A. and H.A.Tucker, North Holland, 1979.

/DATE 74/ : DATE, C. J. An Introduction to Database. Addison Wesley Publishing Company, 1974.

/DATE 83/ : DATE, C. J. :"An Introduction to Database Systems: volume II", Addison-Wesley Publishing Company, 1983.

/DELG 85/ : DELGADO, A.L.N. & MAGALHÃES, L.P.:"Capturing the Semantics of CAD through a Data Model", In Proceedings of 2nd International Conference on Telecommunications and Control, Rio de Janeiro, Dezembro, 1985.

/DELGA 87/ : DELGADO, A.L.N. Banco de dados no contexto de Projeto Auxiliado por Computador. Tese de Mestrado apresentada à Faculdade de Engenharia Elétrica da Universidade Estadual de Campinas.

/GILOI 78/ : GILOI, W. K."Interactive Computer Graphics". Prentice-Hall, New Jersey, 1978.

/HARTELT 86/ : HARTELT, H. I. M. "GAV - Um Gerenciador de Áreas de Visualização Baseado na Norma GKS" Tese de Mestrado apresentada à Faculdade de Engenharia Elétrica da Universidade Estadual de Campinas, dezembro 1986.

/LEBEUX 74 / : LEBEUX, P. "Frame Selection--Systems and Languages for Medical Applications". Office of Medical Information Systems Univ. of California, San francisco. Tecnical Report, July, 1974.

/MAGAL 86/ : MAGALHÃES, LÉO. P. "Computação Gráfica". Editora Universidade Estadual de Campinas-UNICAMP, 1986.

/NEUMANN 82/ : NEUMANN, T. and HORNUNG, C."Consistency and Transactions in CAD Database,"Proceedings of the 8th International Conference on Very Large Data Bases, Mexico City, September, 1982, pp. 181-188.

/NEWMAN 81/ : MEWMAN, M.W. and SPROULL, R.F:"Principles of interactive Computer Graphics" vol.1, Mc Graw-Hill international Book Comp., 1981.

/PHILIPS 84/ : PHILIPS, R and JACKSON, M. "Interfaces in Computing. Vol 2, 1984, pp. 31-43.

/POONEN 78 / : POONEN, G. "CLEAR- A conceptual Language for Entities and Relationship"- Proceedings of International Conference of Management of Data- Italy, August 1978.

/PRESSMAN 82/ : PRESSMAN, R. S. " Software Engineering, A Practitioner's Approach." MC GRAW-HILL. 1982.

/RIC 85/ : RICARTE, I. L. M: " 1o Relatório de atividades".FAPESP, Proc.84/2591-1, Agosto 1985.

/RIC 86/ : RICARTE, I. L. M. "Definição e implementação de um modelo de dados para o Núcleo CORAS-UNICAMP", 3o Relatório de Atividades - FAPESP, Agosto de 1986.

/SENKO 73/ : SENKO, M.E. ALTMAN, E.B. ASTRAHAM, M.M. and FEHDER,P.L. "Data structures and accesing in database systems", IBM Syst.Jour., Vol 12 No 1, 1973, pp. 30-93.

/SILVA 77 / : SILVA, C. P. "Requel: Uma linguagem de Consultas para o Sistema Redas", Dissertação de Mestrado ITA 1977.

/STONE 83/ : STONEBRAKER, M.R. and GUTTMAN, R.A.: "Application of Abstract Data Types and Abstract Indices to CAD Data Bases", Proceedings of IEEE, 1983, pp. 107-113.

/TAYL 76/ : TAYLOR, R.W.and FRANK, R.L."Codasyl-Database Management Systems", Computing Surveys, vol.8, No 1, March,1976.

/ZLOOF 75/ : ZLOOF, M.M . L."Query By Example" Proc.AFIPS 1975 NCC. Vol 44, AFIPS PRESS.

ANEXO A: EXEMPLO DE IMPLEMENTAÇÃO

Este apêndice apresenta o resultado da implementação do exemplo de utilização da LIGG, descrito no capítulo 5. Uma visão geral do interrelacionamento dos módulos que compõem o programa, é fornecida na figura A.1.

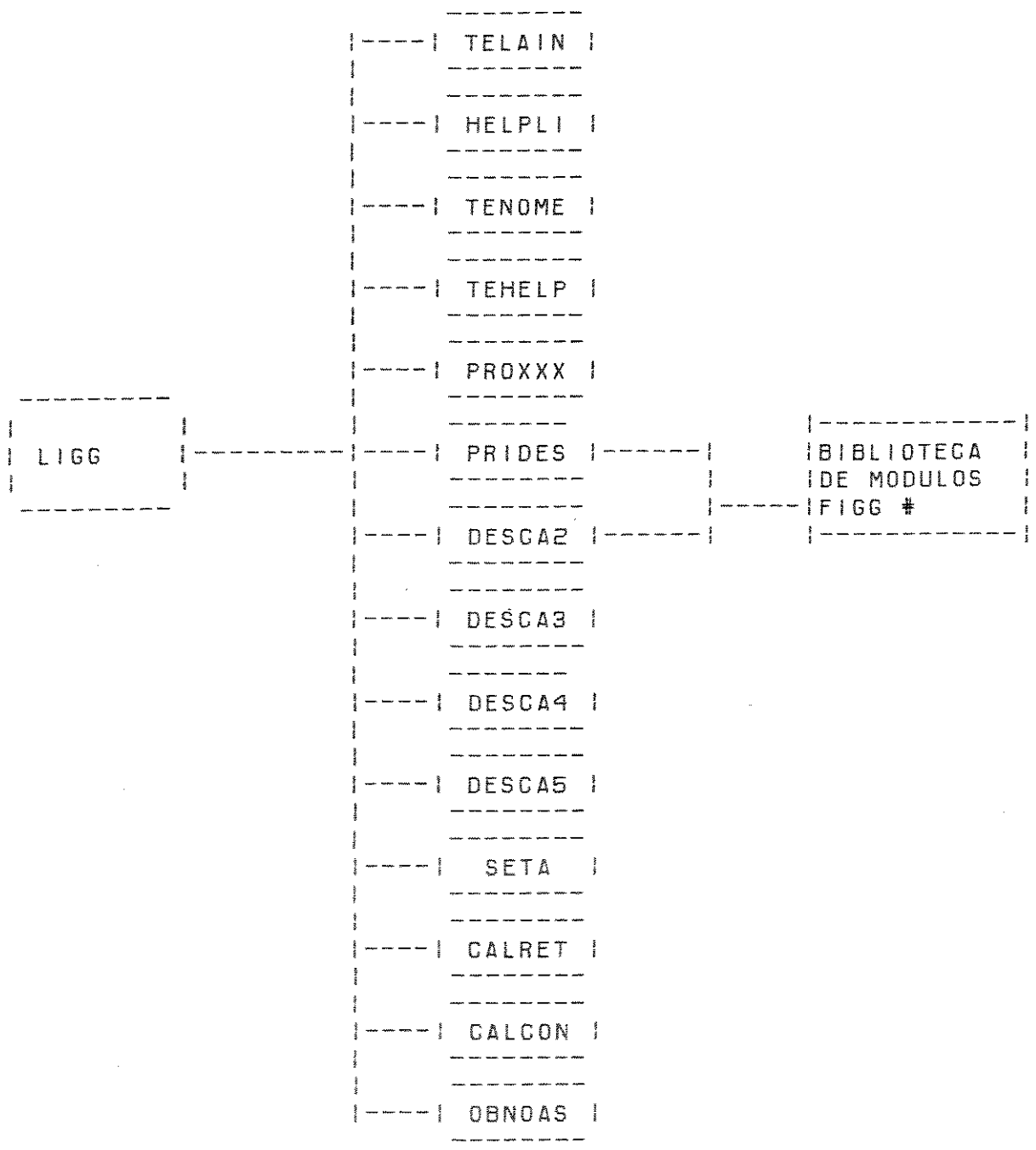


Figura A.1 - Estrutura modular do programa exemplo de implementação da LIGG

```
c***********************************************************************
c*                                                                     *
c*                                                                     *
c*      LIGG -        LINGUAGEM GRAFICA BASEADA EM  GRAFOS             *
c*                                                                     *
c*                                                                     *
c***********************************************************************
c
        implicit none
c
c ---------- inclui arquivos ---------------------------------------
c
        include 'parligg.dat'
        include 'segligg.cmm'
        include 't1ligg.cmm'
        include 't2ligg.cmm'
        include 't3ligg.cmm'
        include 't4ligg.cmm'
        include 't5ligg.cmm'
        include 'vcorlig.cmm'
c
c ----------- declaracao das variaveis locais ----------------------
c
        integer              etas1(7),etas2(5),i,esf(13),sta,ON,OFF,
     *                       pico,sg,tr,etas3(3),chr,OK,NONE,NOPICK,
     *                       ler,CONT,FIRST,SEC,sgg(2),seg(MAXEL),pick,
     *                       lastel,clas,l,inder,nsg,segno(20),flag,
     *                       chave
c
        real                 pox(2),poy(2),posx,posy,xr(2),yr(2),
     *                       xf(5),yf(5),cox(5),coy(5),rx1,ry1,rx2,
     *                       ry2,pretx(2),prety(2)
c
        character*13         tex,nome1,nome2
c
c ------------ inicializacao ---------------------------------------
c
        rentcf=1
        rrelcf=1
        rasscf=1
        ragrcf=1
        rascof=1
        rreagr=1
        do i=1,MAXEL
          nelagr(i)=0
        end do
c
        do i=1,13
          esf(i)=1              ! INDIVIDUAL
        end do
c
        FIRST=0
        SEC=0
        OFF=0
        ON=1
        OK=1
        NONE=0
        NOPICK=0
        CONT=11            ! contador utilizado para nome de seg
c
```

```
c ------- abre o GAV ------------------------------------------
c
      call opgav(55)
c
c ------------ define estacoes de trabalho ----------------------
c
      call dfret(15,5,24,2,0.,.17625,.02,.155,1,2)
      call dfret(16,6,24,2,.17625,.235,.05875,.155,1,2)
      call dfret(17,7,24,2,.0,.17625,.0,.02,1,2)
      call dfret(18,8,24,2,.17625,.235,.02,.05875,1,2)
      call dfret(19,9,24,2,.17625,.235,.0,.02,1,2)
c
c ------------ define  processo visual ---------------------------
c
      do i=1,5
       etas1(i)=14+i        ! ET's as = 11,14,15,16,17,18,19
      end do
      etas1(6)=11
      etas1(7)=14
c
      etas3(1)=12
      etas3(2)=13
      etas3(3)=14
      call dfnpv(1,3,etas3) ! desassocia a ET 11 do PV1
      call dfnpv(2,7,etas1)
c
c ------------ define configuracoes de tela ----------------------
c
      do i=1,5
       etas2(i)=14+i
      end do
      call dfnct(2,5,etas2)
c
c -------- seta PV corrente ---------------------------------------
c
      call setpvc(2)
c
c ------- abre GKS  e traca menu de entrada ----------------------
c
      call xgopks(15)
      call xgsasf(asf)
      call xgopwk(11,11,24)
      call xgacwk(11)
      call telain
      call tenome
      call tehelp
      call xgdawk(11)
c
c -------- obtem opcao desejada ----------------------------------
c
  200 continue
      sta=NONE
      do while (sta.NE.OK)
       call xgrqch(11,41,sta,chr)
      end do
c
      if(chr.EQ.2) then            ! MANIPULAR ESQUEMAS
         go to 400
      else if(chr.EQ.3) then       ! MANIPULAR DADOS
         go to 400
```

```
      else if(chn.EQ.4) then      ! HELP
         call helpli
         go to 100
      else if(chn.EQ.3) then      ! SAIDA
         go to 400
      else if(chn.LT.1.OR.chn.GT.4) then    ! pedir nova opcao
         go to 100
      endif

c
c -------- OPCAO: CRIAR  ----------------------------------------
c
      call proxxx

c
c --------- seta nova CT corrente ------------------------------
c
      call setctc(2)

c
c ------- abre demais ET's -----------------------------------------
c
      if (FIRST.EQ.0)
     *   then
           call xgopuk(15,5,24)
           call xgopuk(16,6,24)
           call xgopuk(17,7,24)
           call xgopuk(18,8,24)
           call xgopuk(19,9,24)

c
c ------------ trace conteudo da area 2 ---------------------------
c
           call xgsuku(16,0.,.61,0.,1.)
           call xgsur(1,0.,18.32,0.,30.)
           call xgsvp(1,0.,.61,0.,1.)
           call xgseln(1)
           call xgacuk(16)
           call desca2
           call xgcauk(16)

c
c ------------ trace conteudo da area 4 ---------------------------
c
           call xgsuku(18,0.,1.,0.,.67)
           call xgsur(1,0.,50.,0.,33.3)
           call xgsvp(1,0.,1.,0.,.67)
           call xgacuk(18)
           call desca4
           call xgcauk(18)

c
c ------------ trace conteudo da area 5 ---------------------------
c
           call xgsuku(19,0.,1.,0.,.34)
           call xgsur(1,0.,50.,0.,17.)
           call xgsvp(1,0.,1.,0.,.34)
           call xgacuk(19)
           call desca5
           call xgcauk(19)

c
c ----------- trace conteudo da area 3 ---------------------------
c
           call xgsuku(17,0.,1.0,0.,0.113)
           call xgsur(1,0.,50.,0.,5.674)
           call xgsvp(1,0.,1.0,0.,0.113)
```

```
        call xgacuk(17)
        call desca3
        call xgcauk(17)
c
      endif
c
c --------- obtem funcao desejada ----------------------------------
c
      SEC=0
 200  continue
      sta=NONE
      do while (sta.NE.OK)
       call xgrqch(15,41,sta,chn)
      end do
c
      if(chn.EQ.2) then              ! REMOVER
         go to 400
      else if(chn.EQ.3) then         ! ALTERAR
         go to 400
      else if(chn.EQ.4) then         ! LISTAR
         go to 400
      else if(chn.EQ.5) then         ! SALVAR
         go to 400
      else if(chn.EQ.6) then         ! FIM
         call xgsvis(1,ON)     ! DEVE RETORNAR AO MENU PRINC
         call setoto(1)
         call xgclru(15,1)
         FIRST=1
         go to 100
      else if(chn.LT.1.OR.chn.GT.6) then ! NOVO CHOICE
         go to 200
      endif
c
c --------- OPCAO: DEFINIR -----------------------------------------
c
c -------- obtem primitiva desejada --------------------------------
c
      sta=NOPICK
      do while (sta.NE.OK)
        call xgrqpk(16,51,sta,sg,picc)
      end do
c
c ------------- escolha p/ def -------------------------------------
c
      if(SEC.EQ.0)
     *      then
              call xgsur(1,0.,39.,0.,30.)
              call xgsvc(1,0.,.39,0.,.3)
              call xgsuku(15,0.,.39,0.,.3)
              call xgsvpi(0,1,1)                     ! SETA PRIORIDADE
              SEC=1
      endif
      call xgseln(1)
      call xgacuk(15)
c
c ------------- agrupamento ----------------------------------------
c
      if(picc.EQ.3) then
c
              lcstcl = 0
```

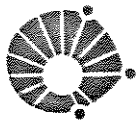
```
      do i=1,MAXEL
        seg(i)=0
      end do

c
      call xgsvis(3,OFF)   ! ESCREVE PROMPT "ENTRE
      call xgsvis(8,ON)    ! NOME" NA ET 17
      call xgrsuk(17)

c
      sta=NOME                                  ! OBTEM NOME
      do while (sta.NE.OK)                      ! DO
       call xgrast(15,61,sta,len,tex) ! AGRUPAMENTO
      end do

c
      agrpo(nagrof) = tex ! ARMAZENA NOME

c
      call xgsvis(8,OFF)   ! ESCREVE PROMPT "APONTE
      call xgsvis(10,ON)   ! ELEMENTOS" NA ET 17
      call xgrsuk(17)

c
      do l=1,MAXEL
                                                 ! OBTEM
        sta=NOPICK                              ! ELEMENTO
        do while(sta.NE.OK)
          call xgrapk(15,51,sta,seg(l),pick)
        end do

c
        if(seg(l).EQ.lastel)
     *     then
            call xgsvis(10,OFF)     !FIM AGRUPAMENTO
            go to 900
        endif

c
        lastel = seg(l)

c
c ------------ armazena dados ------------------------------------
c
        do i = 1,MAXSEG
          if(segro(i).EQ.seg(l))
     *       then
              agrpo(nagrof,nelagr(nagrof)) = nome(i)
              agrpc(nagrof,nelagr(nagrof)) = classe(i)
              relagr(nagrof) = nelagr(nagrof) + 1
              go to 800
          endif
        end do
800     continue
      end do
900   continue

c
c ------- calcula e traca contorno do agrupamento ----------------
c
      call calcon(relagr(nagrof),seg,ccx,ccy)
      call xgcrsg(CONT)
      call xgslw(2)
      call xgpl(5,ccx,ccy)
      call xgsln(1)
      call xgclsg

c
c ---------- armazena dados do agr. na tab. de seg. ---------------
c
      segro(rsegof)=CONT
```

UNICAMP

```
            nome(nsegcf)=tex
            classe(nsegcf)=AGR
            x1(nsegcf)=ccx(1)-.5
            y1(nsegcf)=ccy(1)-.5
            x2(nsegcf)=ccx(3)+.5
            y2(nsegcf)=ccy(3)+.5
            nsegcf=nsegcf+1
            nagrcf = nagrdf + 1
c
          else                         ! NAO E AGRUPAMENTO
c
            call xgsvis(3,OFF)    ! ESCREVE PROMPT "POSICIONE
            call xgsvis(4,ON)     ! CURSOR" NA ET 17
            call xgrsuk(17)
c
c ------------- obtem posicao -------------------------------------
c
  935       continue
            sta=NONE
            do while (sta.NE.OK)
               call xgrolc(15,11,sta,tn,ppx(1),ppy(1))  ! OBTEM POSICAO
            end do
c
            if(picc.EQ.5.OR.picc.EQ.6)
     *         then
                 sta=NONE
                 do while (sta.NE.OK)
                  call xgrolc(15,11,sta,tn,ppx(2),ppy(2))  ! OBTEM NOVA
                 end do                                    ! POSICAO
c
c ----------- obtem elementos da associacao ---------------------
c
                 call obrcas(ppx,ppy,nome1,clas,nome2,pretx,
     *                        prety,flag)
c
c ----------- se associacao e invalida ---------------------------
c
                 if (flag.EQ.1)
     *              then
                      call xgsvis(4,OFF) ! ESCREVE PROMPT: "ASS.
                      call xgsvis(12,ON) ! NAO PERMITIDA"
                      call xgrsuk(17)
c
                      call xgsvis(12,OFF) ! ESCREVE PROMPT: " POS.
                      call xgsvis(4,ON)    ! CURSOR"
                      call xgrsuk(17)
                      go to 935
                 endif
             endif
c
c ----------- calcula retangulo que define vizinhanca ------------
c
            if(picc.NE.5.AND.picc.NE.6)
     *         then
                 call calret(picc,ppx(1),ppy(1),rx1,ry1,rx2,ry2)
            endif
c
c --------- desenha primitiva na ET 15 ---------------------------
c
            call xgpreg(CONT)
```

```
      if(picc.EQ.6)
        then
          call seta(pretx,prety)
        else
          call prices(picc,ppx,ppy)
       endif
c
c -------- escreve prompt ----------------------------------------
c
       call xgsvis(4,CFF)      ! ESCREVE PROMPT "ENTRE NOME"
       call xgsvis(6,CN)       ! NA ET 17
       call xgrswk(17)
       call xgsvis(6,CFF)
c
c ------- obtem NOME da primitiva -------------------------------
c
       sta=NONE
       do while (sta.NE.OK)
         call xgrgst(15,61,sta,len,tex)
       end do
c
c ------ escreve prompt -----------------------------------------
c
       if(picc.NE.5.AND.picc.NE.6)
     *   then
           call xgsvis(4,CN)       ! ESCREVE PROMPT "POSICIONE
           call xgrswk(17)         ! CURSOR" NA ET 17
           call xgsvis(4,CFF)
c
c --------- obtem posicao do texto ---------------------------------
c
           sta=NONE
           do while (sta.NE.OK)
            call xgrqlc(15,11,sta,tr,posx,posy)
           end do
c
c ------ gera texto na posicao obtida ---------------------------
c
           call xgtx(posx,posy,tex)
       endif
c
c ---------- fecha segmento ------------------------------------------
c
       call xgclsg
c
c ---------- armazena dados na estrutura ---------------------------
c
       if(picc.NE.5.AND.picc.NE.6)
     *   then
           segrc(nseg+1) = CONT
           x1(nseg+1) = rx1
           x2(nseg+1) = rx2
           y1(nseg+1) = ry1
           y2(nseg+1) = ry2
           if(picc.EQ.1) then
             classe(nseg+1) = PNT
           else if(picc.EQ.2) then
             classe(nseg+1) = REL
           else if(picc.EQ.4) then
             classe(nseg+1) = ESC
```

```
                endif
                nome(nseqcf) = tex
                nseqcf = nseqcf + 1
            endif
c
            if(picc.EQ.1) then
                ertnc(nertcf) = tex
                nertcf = nertcf + 1
            else if(picc.EQ.2) then
                relnc(nrelcf) = tex
                nrelcf = nrelcf + 1
            else if(picc.EQ.4) then
                chave=0
                do i=1,MAXESQ
                 if(esqnc(i).EQ.tex)
     *             then
                     chave=1
                     go to 940
                 endif
                end do
940             continue
                if(chave.EQ.0)
     *            then          ! NOME ESQUEMA AINDA NAO ARMAZENADO
                    esqnc(nesqcf) = tex
                    nesqcf = nesqcf + 1
                endif
            else if(picc.EQ.5.OR.picc.EQ.6) then
                essnc(nessdf) = tex
                if(clas.EQ.ENT.AND.picc.EQ.6)
     *            then                          ! trace ent trace
                  do i=1,MAXSEQ
                   if(nome(i).EQ.nome1)
     *               then
                       xf(1)=x1(i)+1.5
                       yf(1)=y1(i)+1.5
                       xf(2)=x2(i)-1.5
                       yf(2)=y1(i)+1.5
                       xf(3)=x2(i)-1.5
                       yf(3)=y2(i)-1.5
                       xf(4)=x1(i)+1.5
                       yf(4)=y2(i)-1.5
                       xf(5)=x1(i)+1.5
                       yf(5)=y1(i)+1.5
                       go to 950
                   endif
                  end do
950               continue
                  CONT=CONT+1
                  call xgerseg(CONT)
                  call xgpl(5,xf,yf)
                  call xgclsg
                endif
                essnr(nessdf) = nome1
                essnc(nessdf) = clas
                essne(nessdf) = nome2
                nessdf = nessdf + 1
            endif
        endif
c
c ---------- incrementa contador de segmentos ----------
```



```
c
c UNICAMP
      CONT=CONT+1
      call xgcauk(15)
c
c -------- escreve prompt e obtem nova opcao ----------------------
c
      call xgsvis(3,ON)          ! ESCREVE PROMPT "ESCOLHA FUNCAO
      call xgreuk(17)            ! E PRIMITIVA DESEJADAS" NA ET 17
      go to 200
c
c ---------- FIM ---------------------------------------------------
c
 400  continue
      call xgcluk(11)
      call xgcluk(15)
      call xgcluk(16)
      call xgcluk(17)
      call xgcluk(18)
      call xgcluk(19)
      call xgclks
      call clgev
      end
```

```
c ========== subrotina TELA INICIAL ========================
c
c UNICAMP codigo:criar TELA INICIAL
c
c linguagem: FORTRAN-77
c
c ============================================================
c
      subroutine TELAIN
      implicit none
c
c --------- declaracao de variaveis -------------------------
c
      real              ponx(5),pony(5)
c
      character*8       str1
c
      character*18      str2
c
      character*15      str3
c
      character*7       str4
c
      character*8       str5
c
c --------- inicializacao -----------------------------------
c
      data  porx/.4,.4,.4,.4,.4/
      data  pory/.7,.6,.5,.4,.3/
      str1='1. CRIAR'
      str2='2. MANIPULAR ESQUEMAS'
      str3='3. MANIPULAR DADOS'
      str4='4. HELP'
      str5='5. SAIDA'
c
c ------------------------------------------------------------
c
      call xgcrsg(1)
      call xgtx(porx(1),pory(1),str1)     ! CRIAR
      call xgtx(porx(2),pory(2),str2)     ! MANIPULAR
      call xgtx(porx(3),pory(3),str3)     ! CONSULTAR
      call xgtx(porx(4),pory(4),str4)     ! HELP
      call xgtx(porx(5),pory(5),str5)     ! SAIDA
      call xgclsg
c
      return
      end
```

```
c ========== subrotina helpli ==========================
c
c UNICAMP
c finalidade: apresentar help
c
c linguagem: FORTRAN-77
c
c ========================================================
c
      subroutine helpli
      implicit none
c
c ---------- declaracao de variaveis ----------------------
c
      integer        OFF,ON
c
      character*1    txx1,txx2
c
c ---------- inicializacao --------------------------------
c
      OFF=0
      ON=1
c
c ---------- apresenta primeira parte ---------------------
c
      call xgsvis(1,OFF)
      call xgsvis(9,ON)
      call xgrsuk(11)
c
c ---------- pausa ----------------------------------------
c
      accept 100,txx1
c
c ---------- apresenta segunca parte ----------------------
c
      call xgsvis(9,OFF)
      call xgsvis(11,ON)
      call xgrsuk(11)
c
c ---------- pausa ----------------------------------------
c
      accept 100,txx2
  100 format(a1)
c
c ---------- retorna ao menu principal --------------------
c
      call xgsvis(11,OFF)
      call xgsvis(1,ON)
      call xgrsuk(11)
c
      return
      end
```

```
c ========== subrotina teneme ================================
c
c UNICAMP cade:criar tela com nome esquema e versao
c
c linguagem: FORTRAN-77
c
c ============================================================
c
      subroutine teneme
      implicit none
c
c --------- declaracao de variaveis -------------------------
c
      integer           off
c
      real              posx(2),posy(2)
c
      character*13      str1
c
      character*7       str2
c
c
c --------- inicializacao -----------------------------------
c
      data  posx/.3,.3/
      data  posy/.7,.5/
      str1='NOME ESQUEMA:'
      str2='VERSAO:'
      off = 0
c
c -----------------------------------------------------------
c
      call xgcrsg(2)
      call xgsvis(2,off)
      call xgtx(posx(1),posy(1),str1)     ! NOME ESQUEMA
      call xgtx(posx(2),posy(2),str2)     ! VERSAO
      call xgclsg
c
      return
      end
```

```
c ======= subrotina tehelp ========================================
c
c UNICAMP
c finalidade: criar tela com help
c
c linguagem: FORTRAN-77
c
c ==================================================================
c
      subroutine tehelp
      implicit none
c
c ---------- declaracao de variaveis ------------------------------
c
      integer           OFF
c
      real              ponx(15),pory(15)
c
      character*54      tx1,tx2
      character*18      tx3
      character*9       tx4
      character*13      tx5
      character*10      tx6
      character*12      tx7,tx8
      character*31      tx9
      character*16      tx10,tx11
      character*30      tx12
      character*13      tx13,tx14
      character*33      tx15
c
c ------------- inicializacao ----------------------------------------
c
      OFF=0
c
      data  ponx/.2,.2,.2,.2,.2,.2,.2,.2,.2,.2,.2,.2,.2,
     *           .2,.2/
c
      data pony/.9,.85,.8,.6,.5,.4,.9,.85,.75,.65,.6,.5,.4,
     *          .35,.25/
c
      tx1='ESTA E UMA INTERFACE GRAFICA INTERATIVA DE COMUNICACAO'
      tx2='USUARIO - BANCO DE DADOS. PARA A OPERACAO DO SISTEMA'
      tx3='TEM-SE TRES MODOS:'
      tx4='- CRIACAO'
      tx5='- MANIPULACAO'
      tx6='- CONSULTA'
      tx7='MODO CRIACAO'
      tx8='************'
      tx9='PERMITE DEFINIR A BASE DE DADOS'
      tx10='MODO MANIPULACAO'
      tx11='****************'
      tx12='PERMITE OPERAR A BASE DE DADOS'
      tx13='MODO CONSULTA'
      tx14='************'
      tx15='PERMITE CONSULTAR A BASE DE DADOS'
c
c ------------------------------------------------------------------
c
      call xgcrsg(9)
      call xgsvis(9,OFF)
      call xgtx(ponx(1),pony(1),tx1)
```

UNICAMP

```
      call xgtx(porx(2),pory(2),tx2)
      call xgtx(porx(3),pory(3),tx3)
      call xgtx(porx(4),pory(4),tx4)
      call xgtx(porx(5),pory(5),tx5)
      call xgtx(porx(6),pory(6),tx6)
      call xgclsg
c
      call xgcrsg(11)
      call xgsvis(11,OFF)
      call xgstxc(2)
      call xgtx(porx(7),pory(7),tx7)
      call xgtx(porx(8),pory(8),tx8)
      call xgstxc(1)
      call xgtx(porx(9),pory(9),tx9)
      call xgstxc(2)
      call xgtx(porx(10),pory(10),tx10)
      call xgtx(porx(11),pory(11),tx11)
      call xgstxc(1)
      call xgtx(porx(12),pory(12),tx12)
      call xgstxc(2)
      call xgtx(porx(13),pory(13),tx13)
      call xgtx(porx(14),pory(14),tx14)
      call xgstxc(1)
      call xgtx(porx(15),pory(15),tx15)
      call xgclsg
c
      return
      end
```

```
c ========== subrotina proxxx ==============================
c
c UNICAMP - Finalidade: obter nome da base de dados e versao
c
c  Linguagem: FORTRAN-77
c
c=============================================================
c
      subroutine proxxx
      implicit none
c
      include 't5ligg.cmm'
c
c --------- declaracao de variaveis --------------------------
c
      integer       ler1,ler2,sta,off,on,NONE,OK
c
      real          ppx(2),ppy(2)
c
      character*20  txt1
c
      character*4   txt2
c
      character*1   txx
c
c --------- inicializacao ------------------------------------
c
      OFF=0
      ON=1
      NONE=0
      OK=1
      data ppx/.55,.45/
      data ppy/.7,.3/
c
c -------------------------------------------------------------
c
      call xgacuk(11)
      call xgstxc(2)
      call xgsvis(1,OFF)
      call xgsvis(2,ON)
      call xgrsuk(11)
      call xgsvis(2,OFF)
      call xgselr(0)
c
c ------ obtem nome da base de dados ------------------------
c
  100 continue
      sta=NONE
      do while (sta.NE.OK)
        call xgrgst(11,61,sta,len1,txt1)
      enddo
      if(len1.LT.1.OR.len1.GT.20)
     *    then
            go to 100
          else
            call xgtx(ppx(1),ppy(1),txt1)
      endif
c
c ---------- armazena nome do esquema na tabela 5 ------------
c
```

```
      ESQAC = txt1
c
c UNICAMP---- obtem versao ------------------------------------
c
  200   continue
        sta=NONE
        do while (sta.NE.OK)
           call xgrest(11,61,sta,len2,txt2)
        end do
        if(len2.LT.1.OR.len2.GT.4)
     *    then
            go to 200     ! NOVO PEDIDO
          else
            call xgtx(ppx(2),ppy(2),txt2)
        endif
c
c ----------- armazena versao na tabela 3 --------------------
c
        ESQAT(1) =
c
        accept 450,txx
 450    format(a1)
        call xgstxc(1)
        call xgseln(1)
        call xgcauk(11)
c
c ----------------------------------------------------------------
c
        return
        end
```

```
c ========== subrotina prides ==================================
c
c UNICAMP cede : desenha primitiva na ET 15
c
c linguagem: FORTRAN-77
c
c ================================================================
c
      subroutine prides(pick,posx,posy)
      implicit none
c
c ---------- declaracao de variaveis ----------------------------
c
      integer      pick
c
      real         posx(2),posy(2)
c
c ---------------------------------------------------------------
c
      if(pick.EQ.1) then
         call figg1(posx(1),posy(1))
      else if(pick.EQ.2) then
         call figg2(posx(1),posy(1))
      else if(pick.EQ.3) then
         call figg3(posx(1),posy(1))
      else if(pick.EQ.4) then
         call figg4(posx(1),posy(1))
      else if(pick.EQ.5) then
         call xgpl(2,posx,posy)
      endif
c
      return
      end
```

```
c ==== subrotina desca2 ===================================
c
c finalidade : desenha conteudo area dois
c
c linguagem: FORTRAN-77
c
c =========================================================
c
      subroutine desca2
      implicit none
c
c ---------------------------------------------------------
c
      call xgcrsg(7)
      call xgspki(1)
      call figg1(2.0,22.5)  ! entidade regular
      call xgspki(2)
      call figg2(13.0,21.5) ! relacionamento
      call xgspki(3)
      call figg3(2.0,12.5)  ! agrupamento
      call xgspki(4)
      call figg4(10.0,12.5) ! esquema
      call xgspki(5)
      call figg5(2.0,2.5)  ! ligacao simples
      call xgspki(6)
      call figg6(13.,2.5)  ! ligacao com flecha
      call xgspki(1)
      call xgclsg
c
c ---------------------------------------------------------
c
      return
      end
```

```fortran
c ======= subrotina desca3 ===============================
c
c finalidade: desenhar conteudo da area 3
c
c linguagem: FORTRAN-77
c
c =======================================================
c
      subroutine desca3
      implicit none
c
c --------- declaracao de variaveis ----------------------
c
      integer            OFF
c
      real               posx,posy
c
      character*36       str1
c
      character*16       str2
c
      character*10       str3
c
      character*16       str4
c
      character*24       str5
c
c --------- inicializacao --------------------------------
c
      OFF=0
      data   posx/5./
      data   posy/3./
      str1='APONTE FUNCAO E PRIMITIVAS DESEJADAS'
      str2='POSICIONE CURSOR'
      str3='ENTRE NOME'
      str4='APONTE ELEMENTOS'
      str5='ASSOCIACAO NAO PERMITIDA'
c
c -------------------------------------------------------
c
      call xgcrsg(3)
      call xgtx(posx,posy,str1) ! APONTE FUNCAO E PRIMITIVAS
                                ! DESEJADAS
      call xgclsg
c
      call xgcrsg(4)
      call xgsvis(4,OFF)
      call xgtx(posx,posy,str2) ! POSICIONE CURSOR
      call xgclsg
c
      call xgcrsg(8)
      call xgsvis(8,OFF)
      call xgtx(posx,posy,str3) ! ENTRE NOME
      call xgclsg
c
      call xgcrsg(10)
      call xgsvis(10,OFF)
      call xgtx(posx,posy,str4) ! APONTE ELEMENTOS
      call xgclsg
c
```

```
      call xgcrsg(12)
      call xgsvis(12,OFF)
      call xgtx(pcrx,pcry,str5)
      call xgclsg
c
      return
      end
```

```
c ====== subrotina DESCA4===================================
c
c finalidade:criar cortelco de area 4
c
c linguagem: FORTRAN-77
c
c =========================================================
c
      subroutine desca4
      implicit none
c
c --------- declaracao de variaveis ------------------------
c
      integer          off
c
      real             posx(6),posy(6)
c
      character*9      str1,str2,str3
c
      character*8      str4,str5
c
      character*5      str6
c
c --------- inicializacao ----------------------------------
c
      data  posx/5.,5.,5.,5.,5.,5./
      data  posy/31.5,26.5,21.5,16.5,11.5,6.5/
      str1='1.definir'
      str2='2.remover'
      str3='3.alterar'
      str4='4.listar'
      str5='5.salvar'
      str6='6.fim'
      off=0
c
c ----------------------------------------------------------
c
      call xgcrsg(5)
      call xgtx(posx(1),posy(1),str1)          ! 1.definir
      call xgtx(posx(2),posy(2),str2)          ! 2.remover
      call xgtx(posx(3),posy(3),str3)          ! 3.alterar
      call xgtx(posx(4),posy(4),str4)          ! 4.listar
      call xgtx(posx(5),posy(5),str5)          ! 5.salvar
      call xgtx(posx(6),posy(6),str6)          ! 6.fim
      call xgclsg
c
      return
      end
```

```
c ======= subrotina CESCA5=============================
c
c UNICAMP cade:criar cartelco da area 5
c
c linguagem: FORTRAN-77
c
c ==================================================
c
      subroutine cesca5
      implicit none
c
c --------- declaracao de variaveis ------------------------
c
      integer          cff
c
      real             porx(2),pory(2)
c
      character*14     str1
c
      character*15     str2
c
c --------- inicializacao ----------------------------------
c
      data  porx/5.,5./
      data  pory/14.5,8.5/
      str1='1.MODO ESQUEMA'
      str2='2.MODO ATRIBUTO'
c
c ----------------------------------------------------------
c
      call xcorsg(6)
      call xgstxc(3)
      call xgtx(porx(1),pory(1),str1)     ! 1.MODO ESQUEMA
      call xgstxc(1)
      call xgtx(porx(2),pory(2),str2)     ! 2.MODO ATRIBUTO
      call xgclsg
c
      return
      end
```

```fortran
c ====== subrotina seta ===============================
c
c       dade: calcular pontos e desenhar seta
c
c linguagem: FORTRAN-77
c
c ======================================================
c
      subroutine seta(posx,posy)
      implicit none
c
c -------- declaracao de variaveis --------------------
c
      real posx(2),posy(2),teta,dx,dy,dx1,dy1,px(3),
     *     py(3),teta1
c
c ------------------------------------------------------
c
      teta=ATAN2D((posy(2)-posy(1)),(posx(2)-posx(1)))
      teta1=teta
      if(teta.LT.0)
     *  then
          teta=360+teta
      endif
      dx=COSD(teta-45)
      dy=SIND(teta-45)
      dx1=COSD(135-teta)
      dy1=SIND(135-teta)
c
      px(1)=posx(1)-dx1
      py(1)=posy(1)+dy1
c
      px(2)=posx(1)
      py(2)=posy(1)
c
      px(3)=posx(1)+dx
      py(3)=posy(1)+dy
c
      call xgpl(2,posx,posy)
      call xgpl(3,px,py)
c
      return
      end
```

```
c  ================= subrotina calret ============================
c
c  finalidade :  calcula retangulo que define a vizinhaca
c                de uma primitiva
c
c  linguagem  :  fortran - 77
c
c  ==================================================================
c
      subroutine calret(pic,ptox,ptoy,a1,b1,a2,b2)
      implicit none
c
c  =================== declaracao de variaveis =====================
c
      integer              pic
c
      real                 ptox,ptoy,a1,a2,b1,b2
c
c  ------------------------------------------------------------------
c
c
      if(pic.EQ.1.OR.pic.EQ.4) then
        a1 = ptox - 1.0
        b1 = ptoy - 1.0
        a2 = ptox + 7.0
        b2 = ptoy + 5.0
      else
        a1 = ptox - 4.0
        b1 = ptoy - 1.0
        a2 = ptox + 4.0
        b2 = ptoy + 7.0
      endif
      return
      end
```

```
c ========= SUBROTINA calcon ===================================
c
c finalidade: calcular contorno que envolve elementos de um
c             agrupamento
c
c linguagem: FORTRAN - 77
c
c =============================================================
c
        subroutine calcon(nel,seglis,vx,vy)
        implicit none
c
c -------- inclui arquivos ------------------------------------
c
        include 'parlisg.dat'
        include 'seglis.cmm'
c
c -------- declaracao de variaveis ---------------------------
c
        integer          nel,seglis(MAXEL),i,j
c
        real             vx(5),vy(5),xmin,ymin,xmax,ymax
c
c -----------------------------------------------------------
c
        do j=1,MAXSEG
          if(segno(j).EQ.seglis(1))
     *      then
              xmin=x1(j)
              ymin=y1(j)
              xmax=x2(j)
              ymax=y2(j)
              go to 100
          endif
        end do
 100    continue
c
        do i=2,nel
         do j=1,MAXSEG
          if(seglis(i).EQ.segno(j))
     *      then
             if(x1(j).LT.xmin)
     *         then
                xmin=x1(j)
             endif
             if(x2(j).GT.xmax)
     *         then
                xmax=x2(j)
             endif
             if(y1(j).LT.ymin)
     *         then
                ymin=y1(j)
             endif
             if(y2(j).GT.ymax)
     *         then
                ymax=y2(j)
             endif
             go to 200
          endif
         end do
```

```
      continue
      end cc
C UNICAMP
      vx(1)=xmin
      vy(1)=ymin
      vx(2)=xmax
      vy(2)=ymin
      vx(3)=xmax
      vy(3)=ymax
      vx(4)=xmin
      vy(4)=ymax
      vx(5)=xmin
      vy(5)=ymin
C
      return
      end
```

```
c ================ subrotina obrcas ==========================
c
c UNICAMP cade : obtem nome e classe dos elementos de uma
c                associacao
c
c linguagem  : fortran-77
c
c=============================================================
c
      subroutine obrcas(ppx,ppy,nome1,clas,nome2,porcx,
     *                  porcy,erro)
      implicit none
c
c -------------------- inclui arquivos --------------------------
c
      include 'parligg.cat'
      include 'seglig.cmm'
c
c -------------------- declaracao de variaveis ------------------
c
      integer             i,j,clas,erro,chave1,chave2
      real                ppx(2),ppy(2),porcx(2),porcy(2)
      character*13        nome1,nome2
c
c-------------------------------------------------------------------
c
      chave1=0
      chave2=0
      do i = 1,MAXSEG
       if(ppx(1).GT.x1(i).AND.ppx(1).LT.x2(i).AND.
     *    ppy(1).GT.y1(i).AND.ppy(1).LT.y2(i))
     *  then
          if(classe(i).EQ.2)       ! relacionamento
     *      then
            chave1=1
            nome2 = nome(i)
            porcx(1) = ppx(2)    ! ORDENA
            porcx(2) = ppx(1)
            porcy(1) = ppy(2)
            porcy(2) = ppy(1)
          else
            chave2=1
            nome1= nome(i)
            clas= classe(i)
            porcx(1) = ppx(1)     ! NAO ORDENA
            porcx(2) = ppx(2)
            porcy(1) = ppy(1)
            porcy(2) = ppy(2)
          endif
       endif
      end do
c
      do i = 1,MAXSEG
       if(ppx(1).GT.x1(i).AND.ppx(2).LT.x2(i).AND.
     *    ppy(2).GT.y1(i).AND.ppy(2).LT.y2(i))
     *  then
          if(classe(i).EQ.2)       ! relacionamento
     *      then
            if(chave1.EQ.1)
     *        then                 ! ERRO: 2 REL.
```

UNICAMP

```
                    erro=1
                  else
                    nome2= nome(i)
                    erro=0
                endif
              else
                if(chave2.EQ.1)
*                 then                    ! ERRO: 2 ENT.
                    erro=1
                  else
                    nome1= nome(i)
                    clas= classe(i)
                    erro=0
                endif
            endif
          endif
        end cc
c
        return
        end
```

```
c  ======  subrotina  figg1 ================================
c
c  UNICAMP cade:  tracar primitiva ertidace regular a partir
c               do ponto fornecido
c
c  linguagem: FORTRAN-77
c
c  ===========================================================
c
      subroutine figg1(px,py)
      implicit none
c
c  --------- declaracac de variaveis ----------------------
c
      real           px,py,px1(5),py1(5)
c
c  ---------------------------------------------------------
c
      px1(1)=px
      py1(1)=py
      px1(2)=px+6.0
      py1(2)=py
      px1(3)=px+6.0
      py1(3)=py+4.0
      px1(4)=px
      py1(4)=py+4.0
      px1(5)=px
      py1(5)=py
c
      call xgpl(5,px1,py1)
c
      return
      end
```

```
c ========== subrotina  figg2 ================================
c
c UNICAMP cade: tracar primitiva relacionamento a partir do
c                porto ferrecido
c
c linguagem: FORTRAN-77
c
c ============================================================
c
      subroutine figg2(px,py)
      implicit none
c
c -------- declaracao de variaveis ---------------------------
c
      real           px,py,px1(5),py1(5)
c
c ------------------------------------------------------------
c
      px1(1)=px
      py1(1)=py
      px1(2)=px+3.0
      py1(2)=py+3.0
      px1(3)=px
      py1(3)=py+6.0
      px1(4)=px-3.0
      py1(4)=py+3.0
      px1(5)=px
      py1(5)=py
c
      call xgpl(5,px1,py1)
c
      return
      end
```

```
c ===== subrotina  fig02 ================================
c
c UNICAMP cede: tracar primitiva agrupamento a  partir
c               do ponto fornecido
c
c linguagem: FORTRAN-77
c
c ========================================================
c
      subroutine fig02(px,py)
      implicit none
c
c -------- declaracao de variaveis ----------------------
c
      real          px,py,px1(5),py1(5)
c
c ---------------------------------------------------------
c
      px1(1)=px
      py1(1)=py
      px1(2)=px+6.0
      py1(2)=py
      px1(3)=px+6.0
      py1(3)=py+4.0
      px1(4)=px
      py1(4)=py+4.0
      px1(5)=px
      py1(5)=py
c
      call xgsln(2)
      call xgpl(5,px1,py1)
      call xgsln(1)
c
      return
      end
```

```
c ========== subrotina  figg4 ================================
c
c UNICAMP cade:  tracar primitiva esquema a partir do porto
c                ferrecico
c
c linguagem: FORTRAN-77
c
c ============================================================
c
      subroutine figg4(px,py)
      implicit none
c
c -------- declaracao de variaveis ---------------------------
c
      real           px,py,px1(5),py1(5),px2(3),py2(3),px3(3)
c
c ------------------------------------------------------------
c
      px1(1)=px
      py1(1)=py
      px1(2)=px+6.0
      py1(2)=py
      px1(3)=px+6.0
      py1(3)=py+4.0
      px1(4)=px
      py1(4)=py+4.0
      px1(5)=px
      py1(5)=py
c
      px2(1)=px
      py2(1)=py
      px2(2)=px+1.0
      py2(2)=py+2.0
      px2(3)=px
      py2(3)=py+4.0
c
      px3(1)=px+5.0
      px3(2)=px+5.0
      px3(3)=px+6.0
c
      call xgpl(5,px1,py1)
      call xgpl(3,px2,py2)
      call xgpl(3,px3,py2)
c
      return
      end
```

```
c  ========== subrotina  fig05 ============================
c
c  UNICAMP  finalidade: tracar primitiva de ligacao a partir do porto
c                       forrecido
c
c  linguagem: FORTRAN-77
c
c  ===================================================================
c
      subroutine fig05(px,py)
      implicit none
c
c  -------- declaracao de variaveis ----------------------------
c
      real            px,py,px1(2),py1(2)
c
c  -------------------------------------------------------------------
c
      px1(1)=px
      py1(1)=py
      px1(2)=px+6.0
      py1(2)=py+4.0
c
      call xgpl(2,px1,py1)
c
      return
      end
```

```
c ========== subrotina  figé ==============================
c
c UNICAMP cade: tracar uma seta a partir do ponto fornecido
c                                                          >
c linguagem: FORTRAN-77
c
c ============================================================
c
      subroutine figé(px,py)
      implicit none
c
c -------- declaracao de variaveis ---------------------------
c
      real            px,py,px1(2),py1(2),px2(3),py2(3)
c
c ------------------------------------------------------------
c
      px1(1)=px
      py1(1)=py
      px1(2)=px
      py1(2)=py+4.0
c
      px2(1)=px-1.0
      py2(1)=py+2.0
      px2(2)=px
      py2(2)=py
      px2(3)=px+1.0
      py2(3)=py+2.0
c
      call xgpl(2,px1,py1)
      call xgpl(3,px2,py2)
c
      return
      end
```

```
C  ========== COMMON TABELA1 LIGG ====================
C
C  UNICAMP NOME DA ENTIDADE
C  ENTTI: TIPO DE ENTIDADE
C  ENTAT: ATRIBUTOS DA ENTIDADE
C
C  ================================================================
C
      COMMON /tlligg/  ENTNO(MAXENT), ENTTI(MAXENT),
     *                 ENTAT(MAXENT,MAXATE)
C
      INTEGER          ENTTI
C
      CHARACTER*12     ENTNO
C
      CHARACTER*15     ENTAT
```

```
C     ======== COMMON TABELAS LIGG ====================
C
C  UNICAMP NOME DO RELACIONAMENTO
C  RELTI: TIPO DE RELACIONAMENTO
C  RELAT: ATRIBUTOS DO RELACIONAMENTO
C
C  ================================================================
C
      COMMON /T21IGG/  RELNO(MAXREL), RELTI(MAXREL),
     *                 RELAT(MAXREL,MAXATR)
C
      INTEGER          RELTI
C
      CHARACTER*13     RELNO
C
      CHARACTER*15     RELAT
```

```
C ========== COMMON TABELAS LIGG ====================
C
C UNICAMP NOME DA ASSOCIACAO
C ASSPN: NOME DA PRIMITIVA ASSOCIADA
C ASSPC: CLASSE DA PRIMITIVA ASSOCIADA
C ASSRE: NOME DO RELACIONAMENTO ASSOCIADO
C ASSCA: CARDINALIDADE
C ASSTI: TIPO DE ASSOCIACAO
C
C ====================================================
C
      COMMON /tbligg/  ASSNC(MAXASS), ASSPN(MAXASS),
     *                 ASSPC(MAXASS),ASSRE(MAXASS),
     *                 ASSCA(MAXASS,2),ASSTI(MAXASS)
C
      INTEGER          ASSPC,ASSTI,ASSCA
C
      CHARACTER*13     ASSNC,ASSPN,ASSRE
```

```
C ======= COMMON TABELA4 LIGG ==============================
C
C
C UNICAMP NOME DO AGRUPAMENTO
C AGRTI: TIPO DE AGRUPAMENTO
C AGRPN: NOME DO ELEMENTO
C AGRPC: CLASSE DO ELEMENTO
C AGRRE: REGRAS DO AGRUPAMENTO
C
C ============================================================
C
      COMMON /t4ligg/  AGRNO(MAXAGR), AGRTI(MAXAGR),
     *                 AGRPN(MAXAGR,MAXEL),AGRPC(MAXAGR,MAXEL),
     *                 AGRRE(MAXAGR,MAXRE)
C
      INTEGER          AGRPC,AGRTI,AGRRE
C
      CHARACTER*13     AGRNO,AGRPN
```

```
C     ========== COMMON TABELAS LIGG ====================
C
C UNICAMP NOME DO ESQUEMA
C ESQTI: TIPO DE ESQUEMA
C ESQAT: ATRIBUTOS DO ESQUEMA
C
C ================================================================
C
      COMMON /tbligg/  ESQNO(MAXESQ), ESQTI(MAXESQ),
     *                 ESQAT(MAXESQ,2)
C
      INTEGER          ESQTI, ESQAT
C
      CHARACTER*13     ESQNO
```

```
c  ========== commor primitivas - segmentos liçç ================
c
c  UNICAMPsegno   : nome do segmento
c         nome    : nome daprimitiva
c         classe  :
c         x1      : vizinhança
c         y1      :
c         x2      :
c         y2      :
c
c  ===============================================================
c
          commor /seglig/  segno(MAXSEG),nome(MAXSEG),
      *                    classe(MAXSEG),x1(MAXSEG),
      *                    x2(MAXSEG),y1(MAXSEG),y2(MAXSEG)
c
          inteçer          seçno,classe
c
          real             x1,x2,y1,y2
c
          character*13     nome
```

```
c ========== ARQUIVO DE PARAMETROS  LIGO ========================
c
c UNICAMP: QUANTIDADE MAXIMA DE ENTIDADES
c MAXREL: QUANTIDADE MAXIMA DE RELACIONAMENTOS
c MAXESQ: QUANTIDADE MAXIMA DE ESQUEMAS
c MAXASS: QUANTIDADE MAXIMA DE ASSOCIACOES
c MAXAGR: QUANTIDADE MAXIMA DE AGRUPAMENTOS
c MAXRE:  QUANTIDADE MAXIMA DE REGRAS
c MAXEL:  QUANTIDADE MAXIMA DE ELEMENTOS EM UM AGRUPAMENTO
c MAXATE: QUANTIDADE MAXIMA DE ATRIBUTOS DE UMA ENTIDADE
c MAXATR: QUANTIDADE MAXIMA DE ATRIBUTOS DE UM RELACIONAMENTO
c UNDEF:  VALOR INDEFINIDO
c REGUL:  REGULAR (TIPO DE PRIMITIVA)
c FRACA:  FRACA (TIPO DE PRIMITIVA)
c ENT:    ENTIDADE (CLASSE DE PRIMITIVA)
c REL:    RELACIONAMENTO (CLASSE DE PRIMITIVA)
c AGR:    AGRUPAMENTO (CLASSE DE PRIMITIVA)
c ESQ:    ESQUEMA (CLASSE DE PRIMITIVA)
c ASS:    ASSOCIACAO (CLASSE DE PRIMITIVA)
c MAXSEG: QUANTIDADE MAXIMA DE SEGMENTOS
c
c ================================================================
c
      integer     MAXENT,MAXREL,MAXESQ,MAXASS,MAXAGR,
     *            MAXRE,MAXEL,MAXATR,UNDEF,REGUL,FRACA,
     *            ENT,REL,AGR,ESQ,ASS,MAXSEG,MAXATE
c
      parameter (MAXENT=8,MAXREL=5,MAXESQ=3,MAXASS=8,
     *           MAXAGR=2,MAXRE=4,MAXEL=6,MAXATR=8,
     *           UNDEF=0,REGUL=1,FRACA=2,ENT=1,REL=2,
     *           AGR=3,ESQ=4,ASS=5,MAXSEG=18,MAXATE=8)
```

ANEXO B: DETALHAMENTO DAS ESPECIFICAÇÕES DA LIGG

NOME PROCEDIMENTO: INICIAL

OBJETIVO: Fazer inicializações e chamadas a procedimentos de operação do sistema

PARÂMETROS DE ENTRADA:-

PARÂMETROS DE SAÍDA :-

ERRO:-

NOME PROCEDIMENTO: PROC-ESCREVE-TELA-INICIAL

OBJETIVO: Põe na tela o menu inicial de operação do sistema

PARÂMETROS DE ENTRADA:

PARÂMETROS DE SAÍDA :

ERRO:

NOME PROCEDIMENTO: PROC-TELA-NOME-ESQUEMA

OBJETIVO: Colocar na tela o texto de nome esquema e versão; lê esses valores

PARÂMETROS DE ENTRADA:-

PARÂMETROS DE SAÍDA : nome esquema, versão

ERRO:-

NOME PROCEDIMENTO: PROC-CRIAR

OBJETIVO: Posicionar o cursor no modo esquema e ler entrada do modo

PARÂMETROS DE ENTRADA:-

PARÂMETROS DE SAÍDA :-

ERRO:-

NOME PROCEDIMENTO: PROC-TELA-CRIAR

OBJETIVO: Chamar procedimentos para desenhar o contorno das áreas de operação e os menus

PARÂMETROS DE ENTRADA:-

PARÂMETROS DE SAÍDA :-

ERRO:-

NOME PROCEDIMENTO: PROC-OPERA-MODO-ESQUEMA

OBJETIVO: Selecionar o tipo de função a ser executada no modo esquema

PARÂMETROS DE ENTRADA: nome esquema, versão

PARÂMETROS DE SAÍDA :-

ERRO: . função apontada inválida

NOME PROCEDIMENTO: PROCEDIMENTO DEFINIÇÃO-ESQUEMA

OBJETIVO: Selecionar a classe do elemento em definição

PARÂMETROS DE ENTRADA:-

PARÂMETROS DE SAÍDA :-

ERRO:-

NOME PROCEDIMENTO: DEFINE-ENTIDADE

OBJETIVO: Desenha entidade na área 1 e preenche as tabelas correspondentes

PARÂMETROS DE ENTRADA:-

PARÂMETROS DE SAÍDA :-

ERRO:-

NOME PROCEDIMENTO: DEFINE-RELACIONAMENTO

OBJETIVO: Desenhar relacionamento na área 1 e preencher as tabelas correspondentes

PARÂMETROS DE ENTRADA:-

PARÂMETROS DE SAÍDA :-

ERRO:-

NOME PROCEDIMENTO: DEFINE-ASSOCIAÇÃO

OBJETIVO: Desenhar associação na área 1 e preencher as tabelas correspondentes

PARÂMETROS DE ENTRADA:-

PARÂMETROS DE SAÍDA :-

ERRO:-

NOME PROCEDIMENTO: DEFINE-AGRUPAMENTO

OBJETIVO: Desenhar agrupamento na área 1 e preencher as tabelas correspondentes

PARÂMETROS DE ENTRADA:-

PARÂMETROS DE SAÍDA :-

ERRO:-

NOME PROCEDIMENTO: DEFINE-ELEMENTO-AGRUPAMENTO .

OBJETIVO: Desenhar um agrupamento na área 1 e preencher as tabelas correspondentes

PARÂMETROS DE ENTRADA:-

PARÂMETROS DE SAÍDA :-

ERRO:-

NOME PROCEDIMENTO: DEFINE-ESQUEMA

OBJETIVO: Desenhar um esquema na área 1 e preencher as tabelas correspondentes

PARÂMETROS DE ENTRADA:-

PARÂMETROS DE SAÍDA :-

ERRO:-

NOME PROCEDIMENTO: PROCEDIMENTO-REMOÇÃO-ESQUEMA

OBJETIVO: Selecionar a classe de elemento a remover

PARÂMETROS DE ENTRADA:-

PARÂMETROS DE SAÍDA :-

ERRO:-

NOME PROCEDIMENTO: REMOVE-ENTIDADE

OBJETIVO: Apagar entidade da área 1 e atualizar as tabelas correspondentes

PARÂMETROS DE ENTRADA:-

PARÂMETROS DE SAÍDA :-

ERRO:-

NOME PROCEDIMENTO: TESTE-REMOÇÃO-ENTIDADE

OBJETIVO: Procurar se o elemento a remover existe nas tabelas correspondentes

PARÂMETROS DE ENTRADA:-

PARÂMETROS DE SAÍDA :-

ERRO: . nome objeto a remover não existe

NOME PROCEDIMENTO: REMOVE-RELACIONAMENTO

OBJETIVO: apagar relacionamento da área 1 e atualizar as tabelas correspondentes

PARÂMETROS DE ENTRADA:-

PARÂMETROS DE SAÍDA :-

ERRO:-

NOME PROCEDIMENTO: TESTE-REMOÇÃO-RELACIONAMENTO

OBJETIVO: Procurar se o elemento a remover existe nas tabelas correspondentes

PARÂMETROS DE ENTRADA:-

PARÂMETROS DE SAÍDA :-

ERRO: . objeto a remover não existe

NOME PROCEDIMENTO: REMOVE-ASSOCIAÇÃO

OBJETIVO: Apagar associação da área 1 e atualizar as tabelas correspondentes

PARÂMETROS DE ENTRADA:-

PARÂMETROS DE SAÍDA :-

ERRO:-

NOME PROCEDIMENTO: REMOVE-ELEMENTO-AGRUPAMENTO

OBJETIVO: Apagar elemento de agrupamento da área 1 e atualizar as tabelas correspondentes

PARÂMETROS DE ENTRADA:-

PARÂMETROS DE SAÍDA :-

ERRO:-

NOME PROCEDIMENTO: REMOVE-AGRUPAMENTO

OBJETIVO: Apagar agrupamento da área 1 e atualizar as tabelas correspondentes

PARÂMETROS DE ENTRADA:-

PARÂMETROS DE SAÍDA :-

ERRO:-

NOME PROCEDIMENTO: REMOVE-ESQUEMA

OBJETIVO: Apagar esquema da área 1 e atualizar as tabelas correspondentes

PARÂMETROS DE ENTRADA:-

PARÂMETROS DE SAÍDA :-

ERRO:-

NOME PROCEDIMENTO: PROCEDIMENTO-ALTERAÇÃO-ESQUEMA

OBJETIVO: Selecionar a classe de elemento a alterar

PARÂMETROS DE ENTRADA:-

PARÂMETROS DE SAÍDA :-

ERRO:-

NOME PROCEDIMENTO: ALTERA-ENTIDADE

OBJETIVO: Alterar o nome de uma entidade na área 1 e atualizar tabelas correspondentes

PARÂMETROS DE ENTRADA:-

PARÂMETROS DE SAÍDA :-

ERRO:-

NOME PROCEDIMENTO: ALTERA-RELACIONAMENTO

OBJETIVO: Altera a identificação do relacionamento na área 1
e atualiza as tabelas correspondentes

PARÂMETROS DE ENTRADA:-

PARÂMETROS DE SAÍDA :-

ERRO:-

NOME PROCEDIMENTO: ALTERA-ASSOCIAÇÃO

OBJETIVO: Alterar a identificação da associação e atualizar as tabelas correspondentes

PARÂMETROS DE ENTRADA:-

PARÂMETROS DE SAÍDA :-

ERRO:-

NOME PROCEDIMENTO: ALTERA-AGRUPAMENTO

OBJETIVO: Alterar a identificação do agrupamento e atualizar as tabelas correspondentes

PARÂMETROS DE ENTRADA:-

PARÂMETROS DE SAÍDA :-

ERRO:-

NOME PROCEDIMENTO: ALTERA-ESQUEMA

OBJETIVO: Alterar a identificação do esquema e atualizar as tabelas correspondentes

PARÂMETROS DE ENTRADA:-

PARÂMETROS DE SAÍDA :-

ERRO:-

NOME PROCEDIMENTO: PROC-CRIAR-MODO-ATRIBUTO

OBJETIVO: Põe na tela o menu para a operação de criação dos atributos

PARÂMETROS DE ENTRADA:-

PARÂMETROS DE SAÍDA :-

ERRO:-

NOME PROCEDIMENTO: PROC-APONTA-ELEMENTO

OBJETIVO: selecionar um elemento do esquema em consulta

PARÂMETROS DE ENTRADA:-

PARÂMETROS DE SAÍDA :-

ERRO:-

NOME PROCEDIMENTO: PROC-OPERAR-MODO-ATRIBUTO

OBJETIVO: Seleciona o tipo de função a ser executada no modo atributo

PARÂMETROS DE ENTRADA: nome esquema, versão

PARÂMETROS DE SAÍDA :-

ERRO: Função apontada inválida

NOME PROCEDIMENTO: PROC-DEFINE-ATRIBUTO

OBJETIVO: Preencher as tabelas correspondentes com os atributos para as primitivas entidade, relacionamento ou esquema

PARÂMETROS DE ENTRADA:-

PARÂMETROS DE SAÍDA :-

ERRO:-

NOME PROCEDIMENTO: PROC-REMOVE-ATRIBUTO

OBJETIVO: Remover os atributos das tabelas correspondentes a entidades, relacionamentos ou esquema

PARÂMETROS DE ENTRADA:-

PARÂMETROS DE SAÍDA :-

ERRO:-

NOME PROCEDIMENTO: PROC-ALTERA-ATRIBUTO

OBJETIVO: Alterar os atributos de entidade, relacionamento ou esquema nas tabelas correspondentes

PARÂMETROS DE ENTRADA:-

PARÂMETROS DE SAÍDA :-

ERRO: . nome objeto inválido

NOME PROCEDIMENTO: PROC-ALTERA-DOMÍNIO

OBJETIVO: Alterar os domínios de entidade, relacionamento ou esquema atualizando as tabelas correspondentes

PARÂMETROS DE ENTRADA:-

PARÂMETROS DE SAÍDA :-

ERRO: . nome objeto inválido

NOME PROCEDIMENTO: PROC-MANIPULAR-ESQUEMA

OBJETIVO: Selecionar o modo de operação do sistema no processo de manipulação

PARÂMETROS DE ENTRADA:-

PARÂMETROS DE SAÍDA :-

ERRO: função apontada inválida

NOME PROCEDIMENTO: DESENHA-TELA-ATRIBUTO

OBJETIVO: Desenha as áreas para manipular atributos

PARÂMETROS DE ENTRADA :-

PARÂMETROS DE SAÍDA :-

ERRO :-

NOME PROCEDIMENTO: MODO-ESQUEMA-M

OBJETIVO: Selecionar o tipo de função a ser executada no processo de manipulação

PARÂMETROS DE ENTRADA: nome esquema, versão

PARÂMETROS DE SAÍDA :-

ERRO: . função apontada inválida

NOME PROCEDIMENTO: PROCEDIMENTO-ACRESCENTAR-ESQ-M̃

OBJETIVO: Selecionar a classe de elemento em definição

PARÂMETROS DE ENTRADA:-

PARÂMETROS DE SAÍDA :-

ERRO:-

NOME PROCEDIMENTO: PROC-ACRESCENTAR-ENTIDADE

OBJETIVO: Desenha entidade acrescentada na área 1 e preenche as tabelas correspondentes

PARÂMETROS DE ENTRADA:-

PARÂMETROS DE SAÍDA :-

ERRO: . objeto a acrescentar já existe

NOME PROCEDIMENTO: PROC-ACRESCENTAR-RELACIONAMENTO

OBJETIVO: Desenha relacionamento na área 1 e preencher as tabelas com relacionamentos acrescentados

PARÂMETROS DE ENTRADA:-

PARÂMETROS DE SAÍDA :-

ERRO: . objeto a acrescentar já existe

NOME PROCEDIMENTO: PROC-ACRESCENTAR-ASSOCIAÇÃO

OBJETIVO: Desenha associação na área 1 e preenche as tabelas correspondentes com associações acrescentadas

PARÂMETROS DE ENTRADA:-

PARÂMETROS DE SAÍDA :-

ERRO: objeto a acrescentar já existe

NOME PROCEDIMENTO: PROC-ACRESCENTAR-AGRUPAMENTO

OBJETIVO: Desenha agrupamento na área 1 e preenche tabelas correspondentes

PARÂMETROS DE ENTRADA:-

PARÂMETROS DE SAÍDA :-

ERRO: . objeto a acrescentar já existe

NOME PROCEDIMENTO: PROC-ACRESCENTAR-ELEMENTO-AGRUPAMENTO

OBJETIVO: Desenha elemento do agrupamento na área 1 e preenche as tabelas correspondentes

PARÂMETROS DE ENTRADA:-

PARÂMETROS DE SAÍDA :-

ERRO: . objeto a crescentar já existe

NOME PROCEDIMENTO: PROC-ACRESCENTAR-ESQUEMA

OBJETIVO: Desenha esquema acrescentado na área 1 e preenche as tabelas correspondentes

PARÂMETROS DE ENTRADA:-

PARÂMETROS DE SAÍDA :-

ERRO: . objeto a acrescentar já existe

NOME PROCEDIMENTO: PROCEDIMENTO-REMOVER-ESQ-M

OBJETIVO: Selecionar a classe de elemento a remover

PARÂMETROS DE ENTRADA:-

PARÂMETROS DE SAÍDA :-

ERRO:-

NOME PROCEDIMENTO: PROC-REMOVER-ENTIDADE

OBJETIVO: Apaga entidade da área 1 e atualiza as tabelas correspondentes

PARÂMETROS DE ENTRADA:-

PARÂMETROS DE SAÍDA :-

ERRO: . objeto a remover não existe

NOME PROCEDIMENTO: PROC-REMOVER-RELACIONAMENTO

OBJETIVO: Apagar relacionamento da área 1 e atualizar as tabelas correspondestes

PARÂMETROS DE ENTRADA:-

PARÂMETROS DE SAÍDA :-

ERRO: . objeto a remover não existe

NOME PROCEDIMENTO: PROC-REMOVER-ASSOCIAÇÃO

OBJETIVO: Apagar associação da área 1 e atualizar as tabelas correspondentes

PARÂMETROS DE ENTRADA:-

PARÂMETROS DE SAÍDA :-

ERRO: . objeto a remover não existe

NOME PROCEDIMENTO: PROC-REMOVER-AGRUPAMENTO

OBJETIVO: Apagar agrupamento da área 1 e atualizar as tabelas correspondentes

PARÂMETROS DE ENTRADA:-

PARÂMETROS DE SAÍDA :-

ERRO:-

NOME PROCEDIMENTO: PROC-REMOVER-ELEMENTO-AGRUP

OBJETIVO: Apagar elemento do agrupamento da área 1 e atualizar as tabelas correspondentes

PARÂMETROS DE ENTRADA:-

PARÂMETROS DE SAÍDA :-

ERRO: . objeto a remover não existe

NOME PROCEDIMENTO: PROC-REMOVER-ESQUEMA

OBJETIVO: Apagar esquema da área 1 e atualizar as tabelas correspondentes

PARÂMETROS DE ENTRADA:-

PARÂMETROS DE SAÍDA :-

ERRO: . objeto a remover não existe

NOME PROCEDIMENTO: PROCEDIMENTO-ALTERAR-ESQ-M

OBJETIVO: Seleciona a classe de elemento a alterar

PARÂMETROS DE ENTRADA:-

PARÂMETROS DE SAÍDA :-

ERRO:-

NOME PROCEDIMENTO: PROC-ALTERAR-ENTIDADE

OBJETIVO: Altera identificação da entidade na área 1 e atualiza as tabelas correspondentes

PARÂMETROS DE ENTRADA:-

PARÂMETROS DE SAÍDA :-

ERRO: . objeto inválido

NOME PROCEDIMENTO: PROC-ALTERAR-RELACIONAMENTO

OBJETIVO: Alterar a identificaçào do relacionamento na área 1 e atualizar as tabelas correspondentes

PARÂMETROS DE ENTRADA:-

PARÂMETROS DE SAÍDA :-

ERRO: . objeto inválido

NOME PROCEDIMENTO: PROC-ALTERAR-ASSOCIAÇÃO

OBJETIVO: Alterar identificação da associaçào e atualizar as tabelas correspondentes

PARÂMETROS DE ENTRADA:-

PARÂMETROS DE SAÍDA :-

ERRO: . objeto inválido

NOME PROCEDIMENTO: PROC-ALTERAR-AGRUPAMENTO

OBJETIVO: Alterar identificação do agrupamento na área 1 e atualizar as tabelas correspondentes

PARÂMETROS DE ENTRADA:-

PARÂMETROS DE SAÍDA :-

ERRO: . objeto inválido

NOME PROCEDIMENTO: MODO-ATRIBUTO-M

OBJETIVO: Selecionar o tipo de função a ser executada na manipulação dos atributos

PARÂMETROS DE ENTRADA: nome esquema, versão

PARÂMETROS DE SAÍDA :-

ERRO: . função apontada inválida

NOME PROCEDIMENTO: PROC-ACRESCENTA-ATRIBUTO-ESQ-EXI

OBJETIVO: Preencher as tabelas correspondentes aos atributos acrescentados

PARÂMETROS DE ENTRADA:-

PARÂMETROS DE SAÍDA :-

ERRO: . nome objeto a acrescentar já existe

NOME PROCEDIMENTO: PROC-REMOVE-ATRIBUTO-ESQ-EXI

OBJETIVO: remover os atributos das tabelas correspondentes

PARÂMETROS DE ENTRADA:-

PARÂMETROS DE SAÍDA :-

ERRO: . objeto a remover não existe

NOME PROCEDIMENTO: PROC-ALTERA-ATRIBUTO-ESQ-EXI

OBJETIVO: Alterar os atributos das tabelas correspondentes

PARÂMETROS DE ENTRADA:-

PARÂMETROS DE SAÍDA :-

ERRO: . nome objeto inválido

NOME PROCEDIMENTO: PROC-ALTERA-DOMINIO-ESQ-EXI

OBJETIVO: Atualizar as tabelas correspondentes

PARÂMETROS DE ENTRADA:-

PARÂMETROS DE SAÍDA :-

ERRO: . nome objeto inválido

NOME PROCEDIMENTO: PROCEDIMENTO-SALVAR

OBJETIVO: Enviar a GERPAC a informação correspondente a nome esquema, versão e conteúdo das tabelas

PARÂMETROS DE ESTRADA:-

PARÂMETROS DE SAÍDA :-

ERRO:-

NOME PROCEDIMENTO: PROCEDIMENTO-LISTAR

OBJETIVO: Imprimir na impressora informação contida na área1

PARÂMETROS DE ENTRADA:-

PARÂMETROS DE SAÍDA :-

ERRO:-

NOME PROCEDIMENTO: PROC-MANIPULAR-DADOS

OBJETIVO: Selecionar o tipo de função a ser executada na manipulação dos dados

PARÂMETROS DE ENTRADA: nome esquema, versão

PARÂMETROS DE SAÍDA :-

ERRO:-

NOME PROCEDIMENTO:PROC-TELA-MANIPULA-DADOS

OBJETIVO: Desenhar os contornos das áreas de operação e seus conteúdos

PARÂMETROS DE ENTRADA:-

PARÂMETROS DE SAÍDA :-

ERRO:-

NOME PROCEDIMENTO: PROC-LER-ESQUEMA

OBJETIVO: Por dados do esquema lido nas tabelas correspondentes

PARÂMETROS DE ENTRADA:-

PARÂMETROS DE SAÍDA : nome esquema, versão

erro:-

NOME PROCEDIMENTO: PROC-CONSULTAR

OBJETIVO: Selecionar objeto em consulta e passar o codigo correspondente a GERPAC

PARÂMETROS DE ENTRADA:-

PARÂMETROS DE SAÍDA :-

ERRO: . objeto a consultar não existe

NOME PROCEDIMENTO: PROC-INSERIR

OBJETIVO: Selecionar objeto a inserir e passar o codigo correspondente a GERPAC

PARÂMETROS DE ENTRADA:-

PARÂMETROS DE SAÍDA :-

ERRO: . objeto a inserir inválido

NOME PROCEDIMENTO: PROC-ALTERAR

OBJETIVO: Selecionar o objeto a alterar e passar o codigo correspondente a GERPAC

PARÂMETROS DE ENTRADA:-

PARÂMETROS DE SAÍDA :-

ERRO: . nome objeto inválido

NOME PROCEDIMENTO: PROC-REMOVER

OBJETIVO: Selecionar o objeto a remover e passar o codigo correspondente a GERPAC

PARÂMETROS DE ENTRADA:-

PARÂMETROS DE SAÍDA :-

ERRO: . objeto a remover não existe

NOME PROCEDIMENTO: PROC-CONTEÚDO-ÁREA-HELP

OBJETIVO: Desenhar o contorno e conteúdo da área correspondente ao menu help

PARÂMETROS DE ENTRADA:-

PARÂMETROS DE SAÍDA :-

ERRO:-

ANEXO C: DETALHAMENTO DOS PROCEDIMENTOS DA LIGG

INICIAL

```
. faz Inicialização necessárias nas tabelas
. opção = 0
. enquanto opção for diferente de 5 faça
. chama PROC-ESCREVE-TELA-INICIAL
. coloca cursor na tela
. lê opção
      case (opção)
            1:
              . PROC-TELA-NOME-ESQUEMA
              . PROC-CRIAR

            2:
              . PROC-TELA-NOME-ESQUEMA
              . PROC-MANIPULAR-ESQUEMA

            3:
              . PROC-TELA-NOME-ESQUEMA
              . PROC-MANIPULAR-DADOS

            4:
              . PROC-HELP

. end-enquanto

            5:
              . finalizações necessárias

. fim
```

```
PROC-ESCREVE-TELA-INICIAL


. TEXTO (1. CRIAR)
. TEXTO (2. MANIPULAR ESQUEMA)
. TEXTO (3. MANIPULAR-DADOS)
. TEXTO (4. HELP)
. TEXTO (5. SAÍDA)
. retorna
```

```
PROC-TELA-NOME-ESQUEMA


. TEXTO (NOME ESQUEMA:    )
. TEXTO (VERSÃO       :    )
. lê nome-esquema
. lê versão
. retorna
```

```
PROC-CRIAR


. chama PROC-TELA-CRIAR
. inicializa sistema em modo = modo-esquema
. lê entrada modo
     if entrada modo = modo-atributo
        then
        . chama PROC-CRIAR-MODO-ATRIBUTO
        else
        . chama PROC-OPERA-MODO-ESQUEMA
     end-if
. retorna
```

PROC-TELA-CRIAR

```
*/ ver figura 4.2.2.4 /*
. desenha área 1
. desenha área 2
. desenha área 3
. desenha área 4
. desenha área 5
. desenha conteúdo área 2
      . retângulo (entidade)
      . losango (relacionamento)
      . retângulo tracejado (agrupamento)
      . retângulo (esquema)
      . reta vertical (associação)
      . flecha
. desenha conteúdo área 3
      . TEXTO (APONTE FUNÇÃO E ATRIBUTO)
. desenha conteúdo área 4
      . TEXTO (1. DEFINIR)
      . TEXTO (2. REMOVER)
      . TEXTO (3. ALTERAR)
      . TEXTO (4. LISTAR)
      . TEXTO (5. SALVAR)
      . TEXTO (6. FIM    )
. desenha conteúdo área 5
      . TEXTO (1. MODO ESQUEMA)
      . TEXTO (2. MODO ATRIBUTO)
. retorna
```

```
PROC-OPERA-MODO-ESQUEMA


. usuário aponta função desejada (área 4)
/* usuário tem capacidade de mudar função antes de cada ter-
   mino do enquanto /*
. enquanto função desejada não for salvar ou fim faça
   if função.tipo = definição
      then
        . chama PROCEDIMENTO-DEFINIÇÃO-ESQUEMA
      else
      if função.tipo = remoção
         then
         . chama PROCEDIMENTO-REMOÇÃO-ESQUEMA
         else
         if função.tipo = alteração
            then
             . chama PROCEDIMENTO-ALTERAÇÃO-ESQUEMA
            else
            if função.tipo = listar
               then
                . chama PROCEDIMENTO-LISTAR
                else
                 -
             end-if
          end-if
       end-if
    end-if
. end-enquanto
     if função.tipo = salvar
        then
         . chama PROCEDIMENTO-SALVAR
        else
         -
      end-if
. retorna
```

PROCEDIMENTO-DEFINIÇÃO-ESQUEMA

```
/* usuário tem capacidade de mudar função antes de cada tér-
   mino do enquanto /*
. enquanto   função.tipo  for  definição  o  sistema  lê  o
   elemento.classe (apontamento na área 2)
   if elemento.classe = entidade
      then
       . chama DEFINE-ENTIDADE
      else
      if elemento.classe = relacionamento
         then
          . chama DEFINE-RELACIONAMENTO
         else
         if elemento.classe = associação
            then
             . chama DEFINE-ASSOCIAÇÃO
            else
            if elemento.classe = agrupamento
               then
                . chama DEFINE-AGRUPAMENTO
               else
               if elemento.classe = elemento.agrupamento
                  then
                   . chama DEFINE-ELEMENTO-AGRUPAMENTO
                  else
                  if elemento.classe = esquema
                     then
                      . chama DEFINE-ESQUEMA
                     else
                      -
                   end-if
                end-if
             end-if
          end-if
       end-if
    end-if
. end-enquanto
. retorna
```

DEFINE-ENTIDADE

. desenha entidade correspondente na área 1
. lê nome de entidade e posição na área 1
. escreve nome no elemento gráfico na área 1
. insere entidade.nome na tabela 4.2.1 ENTIDADE.NOME
. insere branco na tabela 4.2.1 ENTIDADE.TIPO
. retorna

DEFINE-RELACIONAMENTO

. desenha relacionamento correspondente na área 1
. lê nome relacionamento e posição na área 1
. escreve nome relacionamento no elemento gráfico na área 1
. insere relacionamento.nome na tabela 4.2.2 RELACIONAMENTO.
  NOME
. insere branco na tabela 4.2.2 RELACIONAMENTO.TIPO
. retorna

DEFINE-ASSOCIAÇÃO

```
. desenha associação na área 1
. lê nome da associação e elementos
  if entidade.tipo = branco ou regular
     then
       . insere associação.tipo na tabela 4.2.1 ENTIDADE.TIPO
     else
       -
   end-if
   if relacionamento.tipo = branco ou regular
      then
        . insere associação.tipo na tabela 4.2.2 RELACIONA-
          MENTO.TIPO
      else
       -
    end-if
. insere associação.nome na tabela 4.2.3 ASSOCIAÇÃO.NOME
. insere associação.cardinalidade na tabela 4.2.3 ASSOCIA-
  ÇÃO.CARDINALIDADE
. insere associação.tipo na tabela 4.2.3 ASSOCIAÇÃO.TIPO
. insere entidade.nome na tabela 4.2.3 ASSOCIAÇÃO.ENT-NOME
. insere relacionamento.nome na tabela 4.2.3 ASSOCIAÇÃO.
  NOME-REL
. retorna
```

DEFINE AGRUPAMENTO

. lê elementos do agrupamento
. desenha agrupamento correspondente na área 1
. lê nome agrupamento
. escreve nome agrupamento no elemento gráfico na área 1
. insere agrupamento.nome na tabela 4.2.4 AGRUPAMENTO.NOME
. insere brancos na tabela 4.2.4 AGRUPAMENTO.TIPO
. insere agrupamento.elemento.nome na tabela 4.2.4 AGRUPAMENTO.ELEMENTO.NOME
. insere agrupamento.elemento.classe na tabela 4.2.4 AGRUPAMENTO.ELEMENTO.CLASSE
. retorna

DEFINE-ELEMENTO-AGRUPAMENTO

. desenha elemento do agrupamento na área 1
. lê nome do agrupamento
. escreve nome do agrupamento no elemento gráfico na área 1
. insere agrupamento.elemento.nome na tabela 4.2.4 AGRUPAMENTO.ELEMENTO.NOME
. insere agrupamento.elemento.classe na tabela 4.2.4 AGRUPAMENTO.ELEMENTO.CLASSE
. retorna

DEFINE-ESQUEMA

. desenha esquema correspondente na área 1
. lê nome de esquema
. escreve nome do esquema no elemento gráfico na área 1
. insere esquema.nome na tabela 4.2.5 ESQUEMA.NOME
. insere branco na tabela 4.2.5 ESQUEMA.TIPO
. insere versão.tipo na tabela 4.2.5 ESQUEMA.ATRIBUTO
. insere expansão.tipo na tabela 4.2.5 ESQUEMA.ATRIBUTO
. retorna

```
PROCEDIMENTO-REMOÇÃO-ESQUEMA


*/ usuário tem capacidade de mudar função antes de cada tér-
   mino do enquanto /*
. enquanto função.tipo for remoção o sistema lê o elemento.
  classe
    if elemento.classe = entidade
       then
        . chama REMOVE-ENTIDADE
       else
       if elemento.classe = relacionamento
          then
           . chama REMOVE-RELACIONAMENTO
          else
          if elemento.classe = associação
             then
              . chama REMOVE-ASSOCIAÇÃO
             else
             if elemento.classe = agrupamento
                then
                 . chama REMOVE-AGRUPAMENTO
                else
                if elemento.classe = elemento.agrupamento
                   then
                    . chama REMOVE-ELEMENTO-AGRUPAMENTO
                   else
                   if elemento.classe = esquema
                      then
                       . chama REMOVE-ESQUEMA
                      else
                       -
                   end-if
                end-if
             end-if
          end-if
       end-if
    end-if
. end-enquanto
. retorna
```

REMOVE-ENTIDADE

. verifica consistência dos parâmetros de entrada e sinaliza flag de erro
  if flag de erro = ok
     then
       . chama TESTE-REMOÇÃO-ENTIDADE
     else
       -
  end-if
. o sistema pede confirmação se ainda deseja fazer a remoção
. a partir da confirmação, o usuário apaga o grafo, conforme definido em tabela 4.2.15, tabela 4.2.16 e tabela 4.2.17
. retorna

TESTE-REMOÇÃO-ENTIDADE

```
. procura   nas  tabelas  ENTIDADE, AGRUPAMENTO,  ESQUEMA,
  por  entidade.nome
  if entidade.nome  não  existe nas tabelas ENTIDADE.NOME,
     AGRUPAMENTO.NOME, ESQUEMA.NOME
     then
      . erro
     else
      . coloca entidade.nome na tabela 4.2.15,sem duplicação
      . coloca    entidade.classe    na   tabela   4.2.15,   sem
        duplicação
      . procura na tabela ASSOCIAÇÃO por entidade.nome
        if entidade.nome existe na tabela ASSOCIAÇÃO
           then
            . coloca na tabela 4.2.16   toda a ssociação que
              esteja associada a entidade.nome,sem duplicação
            . coloca    na    tabela    INTERNA    RELACIONAMENTO
              todo  relacionamento  que  esteja  associado  a
              associação.nome  e  que  o   relacionamento.tipo
              seja fraco
           else
            -
        end-if
     . enquanto tabela INTERNA-RELACIONAMENTO seja dif.de { }
     . chama TESTE-REMOÇÃO-RELACIONAMENTO
     . end-enquanto
  end-if
. retorna
```

REMOVE-RELACIONAMENTO

. verifica consistência dos parâmetros de entrada e sinaliza flag de erro.
  if flag de erro = ok
     then
       . chama TESTE-REMOÇÃO-RELACIONAMENTO
     else
       -
  end-if
. o sistema pede confirmação se ainda se deseja fazer a remoção.
. a partir da confirmaçao o usuario apaga o grafo conforme definido nas tabelas 4.2.15, 4.2.16 e 4.2.17.
. retorna

TESTE-REMOÇÃO-RELACIONAMENTO

```
. procura na tabela RELACIONAMENTO por relacionamento.nome
. if relacionamento.nome não existe na tabela
     RELACIONAMENTO.NOME
     then
       . ERRO
     else
       . coloca relacionamento.nome na tabela 4.2.17, sem
         duplicação
       . procura na tabela ASSOCIAÇÃO por relacionamen-
         to.nome
         if relacionamento.nome existe na tabela ASSOCIAÇÃO
            then
              . coloca na tabela 4.2.16 toda associação que
                esteja associada a relacionamento.nome, sem
                duplicação
              . coloca na tabela interna ENTIDADE toda
                entidade que esteja associada a associa-
                ção.nome e que associação.tipo seja fraco
         else
          -
         end-if
       . enquanto tabela interna ENTIDADE seja
         diferente de { }
       . chama TESTE-REMOÇÃO-ENTIDADE
       . end-enquanto
.  end-if
. retorna
```

REMOVE-ASSOCIAÇÃO

```
. verifica   consistência   dos   parâmetros   de   entrada e
  sinaliza flag de erro
  if flag de erro = ok
     then
       . procura  na tabela ASSOCIAÇÃO por associação.nome
         if associação.nome   nao   existe   na      tabela
            ASSOCIAÇÃO.NOME
            then
             . erro
            else
             . procura na tabela  de ASSOCIAÇÃO  entidades
               e  relacionamentos   que    participam   de
               associação.nome
             . cria tabela 4.2.18 ASSOC. contendo todas as
               associacões para  ENTIDADE  exceto associa-
               ção.nome    e    tabela 4.2.19      ASSOCIAÇÃO
               contendo todas as associações para RELACIO-
               NAMENTO  exceto associação.nome
               if tabela 4.2.18 = { }
                  then
                    ENTIDADE.TIPO = branco
                  else
                  if tabela 4.2.18 diferente de { } e todas
                     regulares
                  then
                    ENTIDADE.TIPO = regular
                  else
                  if tabela 4.2.18 diferente de { } e existe
                     fraca
                     then
                       ENTIDADE.TIPO = fraca
                end-if
                if tabela 4.2.19 = { }
                   then
                     RELACIONAMENTO.TIPO = branco
                   else
                     if tabela 4.2.19 diferente de {  }  e
                        todas  regulares
                        then
                          RELACIONAMENTO.TIPO = regular
                        else
```

```
                        if tabela 4.2.19 diferente de {   }
                           e  existe fraca
                           then
                             RELACIONAMENTO.TIPO = fraca
                        end-if
               end-if
```

. o sistema pede confirmação se ainda se deseja fazer a remoção
. a partir da confirmação se remove associação.nome da tabela 4.2.18
  else
   -
  end-if
. retorna

REMOVE-ELEMENTO-AGRUPAMENTO

```
. verifica    consistência  dos   parâmetros de  entrada e
  sinaliza flag de erro
  if flag de erro = ok
     then
      . procura     na     tabela     AGRUPAMENTO          por
        agrupamento.elemento.nome
        if agrupamento.elemento.nome   não   existe      na
           tabela  AGRUPAMENTO.ELEMENTO.NOME
           then
            . erro
           else
            . coloca agrupamento.elemento.nome  na  tabela
              4.2.15, sem duplicacão
            . sistema pede confirmação se ainda se  deseja
              fazer a remoção
            . a partir da   confirmação  o  usuário  apaga
              o grafo
         end-if
      else
       -
  end-if
. retorna
```

REMOVE-AGRUPAMENTO

. chama REMOVE-ENTIDADE
. verifica remoção para tabela 4.2.4
. retorna

REMOVE-ESQUEMA

. chama REMOVE-ENTIDADE
. verifica remoção para tabela 4.2.5
. retorna

```
PROCEDIMENTO-ALTERAÇÃO-ESQUEMA


*/ usuário tem capacidade de mudar  função  antes de  cada
   término do enquanto /*
. enquanto função.tipo for alteração o sistema lê o  ele-
  mento.classe
  if elemento.classe = entidad
     then
      . chama ALTERA-ENTIDADE
     else
     if elemento.classe = relacionamento
        then
         . chama ALTERA-RELACIONAMENTO
        else
        if elemento.classe = associação
           then
            . chama ALTERA-ASSOCIAÇÃO
           else
           if elemento.classe = agrupamento
              then
               . chama ALTERA-AGRUPAMENTO
              else
              if elemento.classe = esquema
                 then
                  . chama ALTERA-ESQUEMA
                 else
                  -
               end-if
            end-if
         end-if
      end-if
   end-if
. end enquanto
. retorna
```

ALTERA-ENTIDADE

```
. verifica    consistência  dos   parâmetros  de  entrada e
  sinaliza  flag de erro
  if flag de erro = ok
     then
      if novo.entidade.nome existe na tabela ENTIDADE.NOME
         ou   RELACIONAMENTO.NOME   ou    AGRUPAMENTO.NOME
         ou  ESQUEMA.NOME
         then
          . mensagem "nome entidade ja existe"
         else
          . sustitue antigo.entidade.nome por nova.entida-
            de.nome  na tabela ENTIDADE.NOME
          . altera identificação da entidade na tela
            if antigo.entidade.nome  existe   na    tabela
               AGRUPAMENTO.ELEMENTO.NOME e     ASSOCIAÇÃO.
               NOME-ENT
               then
                . sustitui   associação.nome-ent       por
                  nova.entidade.nome
               else
                -
             end-if
       end-if
   else
    -
. end-if
. retorna
```

```
ALTERA-RELACIONAMENTO

. verifica   consistência   dos   parâmetros   de   entrada e
  sinaliza flag de erro
  if flag de erro = ok
     then
      if novo.relacionamento.nome   existe    na    tabela
         RELACIONAMENTO.NOME   ou    ENTIDADE.NOME      ou
         AGRUPAMENTO.NOME ou ESQUEMA.NOME
         then
          . mensagem "nome relacionamento ja existe"
         else
          . sustitui    antigo.relacionamento.nome     por
            novo.relacionamento.nome na  tabela RELACIONA-
            MENTO.NOME
          . usuário altera identificação do relacionamento
            na área 1
            if antigo.relacionamento.nome     existe    na
               tabela AGRUPAMENTO.ELEMENTO.NOME e ASSOCIA-
               ÇÃO.NOME-ENT
               then
                . sustitui    associação.nome.rel       por
                  novo.relacionamento.nome
                  else
                   -
            end-if
       end-if
       else
        -
. end-if
. retorna
```

```
ALTERA-ASSOCIAÇÃO

. verifica  consistência  dos  parâmetros  de  entrada  e
  sinaliza flag de erro
  if flag de erro = ok
     then
      if nova.associação.nome  existe  na  tabela  ASSO-
         CIAÇÃO.NOME
         then
          if associação.nome-rel          associado          a
             antiga.associação.nome=associação.nome-rel  da
             nova.associação.nome
             then
              . erro
             else
              . sustitui     antiga.associação.nome     por
                nova.associação.nome   na   tabela ASSOCIA-
                ÇÃO.NOME
              . usuário altera identificação da  associação
                na área 1
           end-if
          else
           -
     end-if
    else
     -
. end-if
.retorna
```

ALTERA-AGRUPAMENTO

```
. verifica    consistência  dos   parâmetros  de   entrada e
  sinaliza  flag de erro
  if flag de erro = ok
     then
      if novo.agrupamento.nome   existe  na   tabela   ENTI-
         DADE.NOME   ou   RELACIONAMENTO.NOME   ou    AGRUPA-
         MENTO.NOME ou ESQUEMA.NOME
         then
          . mensagem "nome agrupamento ja existe"
         else
          . sustitui       antigo.agrupamento.nome        por
            novo.agrupamento.nome    na   tabela    AGRUPA-
            MENTO.NOME
          . usuário  altera  identificação  do  agrupamento
            na área 1
            if antigo.agrupamento.nome  existe  na   tabela
               AGRUPAMENTO.ELEMENTO.NOME    ou     ASSOCIA-
               ÇÃO.NOME-ENT
               then
                . sustitui  antigo.agrupamento.nome     por
                  novo.agrupamento.nome  na  tabela AGRUPA-
                  MENTO.NOME
               else
                -
             end-if
        end-if
    else
    -
. end-if
```

ALTERA-ESQUEMA

```
. verifica   consistência   dos  parâmetros  de   entrada  e
  sinaliza flag de erro
. if flag de erro = ok
     then
      if novo.esquema.nome  existe na  tabela  ENTIDADE.NOME
      ou RELACIONAMENTO.NOME  ou  AGRUPAMENTO.NOME ou ESQUE-
      MA.NOME
      then
        . mensagem "nome esquema ja existe"
      else
        . sustitui antigo.esquema.nome  por novo.esquema.nome
          na tabela ENTIDADE.NOME
        . usuário altera identificação do esquema na área 1
          if antigo.esquema.nome    existe    na      tabela
             AGRUPAMENTO.ELEMENTO.NOME     ou      ASSOCIA-
             ÇÃO.NOME-ENT
             then
               . sustitui       associação.nome-ent       por
                 novo.esquema.nome
           else
             -
        end-if
    end-if
   else
    -
. end-if
```

```
PROC-CRIAR-MODO-ATRIBUTO

*/ ver figura 4.2.17 /*
. apaga conteúdo área 2
. apaga conteúdo área 3
. desenha conteúdo área 2
      . TEXTO (DOMÍNIO)
. desenha contorna área 6
. desenha conteúdo área 6
      . TEXTO (INTEIRO)
      . TEXTO (REAL    )
      . TEXTO (CARACTER)
. desenha conteúdo área 3
. TEXTO (APONTE ELEMENTO)
. chama PROC-APONTA-ELEMENTO
. chama PROC-OPERA-MODO-ATRIBUTO
. retorna
```

PROC-APONTA-ELEMENTO

. enquanto usuário aponta elemento na área 1
. usuário aponta elemento
. lê elemento
. end-enquanto
. torna invisíves elementos da área 1 não apontados
. apresenta na área 1 atributos já existentes
. apaga área 3
. retorna

```
PROC-OPERA-MODO-ATRIBUTO


*/ usuário tem  capacidade de  mudar  função  antes  de cada
   término do enquanto */
. enquanto função desejada não for salvar ou fim faça
  if função.tipo = definição
        then
         . chama PROC-DEFINE-ATRIBUTO
        else
        if função.tipo = remoção
           then
            . chama PROC-REMOVE-ATRIBUTO
           else
           if função.tipo = alteração
              then
               . chama PROC-ALTERA-ATRIBUTO
               . chama PROC-ALTERA-DOMINIO
              else
              if função.tipo = listar
                 then
                  . chama PROCEDIMENTO-LISTAR
                 else
                   -
              end-if
           end-if
        end-if
   end-if
. end-enquanto
  if função.tipo = salvar
     then
      . chama PROCEDIMENTO-SALVAR
     else
      -
  end-if
. retorna
```

```
PROC-DEFINE-ATRIBUTO

*/ usuário tem capacidade de mudar função antes de cada
   término do enquanto /*
. enquanto função seja define.atributo o sistema lê nome
  atributo
/ texto área 3 (ATRIBUTO:)
  if primitiva.classe = entidade
     then
      . enquanto existam atributos por definir e
        primitiva.classe = entidade faça
        if atributo.nome ja existe na tabela ENTIDADE
           then
            . erro
           else
            . define atributo.nome na tabela
              ENTIDADE.ATRIBUTO.NOME
            . define domínio.nome na tabela
              ENTIDADE.ATRIBUTO.DOMÍNIO
         end-if
      end-enquanto
  end-if
  if primitiva.classe = esquema
     then
      . enquanto existam atributos por definir e
       primitina.classe = esquema faça
       if atributo.nome ja existe na tabela ESQUEMA
          then
           . erro
          else
           . define atributo.nome na tabela
             ESQUEMA.ATRIBUTO.NOME
           . define atributo.domínio na tabela
             ESQUEMA.ATRIBUTO.DOMÍNIO
        end-if
      end-enquanto
      if primitiva.classe = relacionamento
         then
          . enquanto existam atributos por definir e
            primitiva.classe = relacionamento faça
            if atributo.nome ja existe na tabela
               RELACIONAMENTO
               then
                . erro
               else
                . define atributo.nome na tabela
                  RELACIONAMENTO.ATRIBUTO.NOME
                . define atributo.domínio na tabela
                  RELACIONAMENTO.ATRIBUTO.DOMÍNIO
             end-if
          end-enquanto
            end-enquanto
.retorna
```

```
PROC-REMOVE-ATRIBUTO

*/ usuário tem capacidade de mudar função antes de cada
   término do enquanto /*
. enquanto função seja remove.atributo o sistema lê nome
  atributo
  if primitiva.classe = entidade
     then
      . enquanto existam atributos por remover e
        primitiva.classe = entidade faça
        if atributo.nome não existe na tabela ENTIDADE
           then
            . erro
           else
            . apaga atributo.nome na tabela
              ENTIDADE.ATRIBUTO.NOME
            . apaga atributo.domínio na tabela
              ENTIDADE.ATRIBUTO.DOMÍNIO
         end-if
       end-enquanto
    if primitiva.classe = esquema
       then
        . enquanto existam atributos por remover e
          primitiva.classe = esquema faça
          if atributo.nome não existe na tabela ENTIDADE
             then
              . erro
             else
              . apaga atributo.nome na tabela
                ENTIDADE.ATRIBUTO.NOME
              . apaga atributo.domínio na tabela
                ENTIDADE.ATRIBUTO.DOMÍNIO
           end-if
         end-enquanto
         if primitiva.classe = relacionamento
            then
             . enquanto existam atributos por remover e
               primitiva.classe = relacionamento faça
               if atributo.nome não existe na tabela
                  RELACIONAMENTO
                  then
                   . erro
                  else
                   . apaga atributo.nome da tabela
                     RELACIONAMENTO.ATRIBUTO.NOME
                   . apaga atributo.domínio da tabela
                     RELACIONAMENTO.ATRIBUTO.DOMÍNIO
                end-if
             end-enquanto
         else
   end-enquanto
. retorna
```

```
PROC-ALTERA-ATRIBUTO

*/ usuário  tem capacidade de mudar função antes de cada
   término do enquanto /*
. enquanto função seja altera.atributo o sistema lê nome
  atributo
  if primitiva.classe = entidade
     then
      . enquanto existam atributos por alterar e
        primitiva.classe = entidade faça
        if novo.atributo.nome existe na tabela ENTIDADE
           then
            . mensagem "nome atributo ja existe"
           else
            . sustitui antigo.atributo.nome por
              novo.atributo.nome na tabela
              ENTIDADE.ATRIBUTO.NOME
         end-if
       end-enquanto
    end-if
    if primitiva.classe = esquema
       then
        . enquanto existam atributos por alterar e
          primitiva.classe = esquema faça
          if novo.atributo.nome existe na tabela ENTIDADE
             then
              . mensagem "nome atributo ja existe"
             else
              . sustitui antigo.atributo.nome por
                novo.atributo.nome na tabela
                ENTIDADE.ATRIBUTO.NOME
          end-if
        end-enquanto
    end-if
   if primitiva.classe = relacionamento
      then
       . enquanto existam atributos por alterar e
         primitiva.classe = relacionamento faça
         if novo.atributo.nome existe na tabela
            RELACIONAMENTO
            then
             . mensagem "nome atributo ja existe"
            else
             . sustitui antigo.atributo.nome por
               novo.atributo.nome na tabela
               RELACIONAMENTO.ATRIBUTO.NOME
         end-if
       end-enquanto
    end-if
  end-enquanto
. retorna
```

```
PROC-ALTERA-DOMÍNIO


*/ usuário tem capacidade de mudar função antes de cada
   término do enquanto /*
 . enquanto função seja   altera.domínio o sistema lê nome
   do domínio
   if primitiva.classe = entidade
      then
       . enquanto   existam   atributos   por   alterar   e
         primitiva.classe = entidade faça
         if novo.atributo.domínio  existe na tabela ENTIDADE
            then
             . mensagem " domínio do atributo ja existe
            else
             . sustitui       antigo.atributo.domínio      por
               novo.atributo.domínio       na          tabela
               ENTIDADE.ATRIBUTO.DOMÍNIO
          end-if
       end-enquanto
    end-if
    if primitiva.classe = esquema
       then
        . enquanto   existam   atributos   por   alterar   e
          primitiva.classe = esquema faça
          if novo.atributo.domínio existe na tabela ENTIDADE
             then
              . mensagem " nome domínio ja existe"
             else
              . sustitui       antigo.atributo.domínio     por
                novo.atributo.domínio       na          tabela
                ENTIDADE.ATRIBUTO.DOMÍNIO
           end-if
        end-enquanto
   end-if
   if primitiva.classe = relacionamento
      then
       . enquanto  existam  atributos  por  alterar  e
         primitiva.classe = relacionamento faça
         if novo.atributo.domínio     existe    na    tabela
            RELACIONAMENTO
            then
             . mensagem " nome domínio ja existe
               else
             . sustitui     antigo.atributo.domínio        por
               nove.atributo.domínio       na          tabela
               RELACIONAMENTO.ATRIBUTO.DOMÍNIO
          end-if
      end-enquanto
  end-if
  end-enquanto
. retorna
```

PROC-MANIPULAR-ESQUEMA

```
. chamar PROC-TELA-CRIAR
. lê    dados   via    GERPAC  fornecendo  esquema  e  versão,
  preenchendo as tabelas correspondentes
. lê entrada
  if operação.modo = modo-esquema
     then
       . chama MODO-ESQUEMA-M
     else
     if operação.modo = modo-atributo
        then
          . chama DESENHA-TELA-ATRIBUTO
          . chamar MODO-ATRIBUTO-M
        else
          -
      end-if
   end-if
. retorna
```

DESENHA-TELA-ATRIBUTO

```
. desenha área 1
. desenha área 2
. desenha área 3
. desenha área 4
. desenha área 5
. desenha conteúdo área 2
      . TEXTO(DOMÍNIO)
      . desenha contorno área 6
      . desenha conteúdo área 6
      . TEXTO(INTEIRO)
      . TEXTO(REAL)
      . TEXTO(CARACTER)
. desenha conteúdo área 3
      . TEXTO(APONTE PRIMITIVA)
. desenha conteúdo área 4
      . TEXTO(1.DEFINIR)
      . TEXTO(2.REMOVER)
      . TEXTO(3.ALTERAR)
      . TEXTO(4.LISTAR)
      . TEXTO(5.SALVAR)
      . TEXTO(6.FIM)
. desenha conteúdo área 5
      . TEXTO(1.MODO ESQUEMA)
      . TEXTO(2.MODO ATRIBUTO)
. retorna
```

MODO-ESQUEMA-M

```
*/ usuário   tem   capacidade  de mudar função antes de cada
   término d enquanto /*
. enquanto  função  desejada  não  for  salvar  ou  fim faça
  if função.tipo = acrescentar
     then
      . chama PROCEDIMENTO-ACRESCENTAR-ESQ-M
     else
     if função.tipo = remoção
        then
         . chama PROCEDIMENTO-REMOVER-ESQ-M
        else
        if função.tipo = alteração
           then
            . chama PROCEDIMENTO-ALTERAR-ESQ-M
           else
           if função.tipo = listar
              then
               . chama PROCEDIMENTO-LISTAR-ESQ-M
              else
               -
           end-if
        end-if
     end-if
  end-if
. end-enquanto
  if função.tipo = salvar
     then
      . chama PROCEDIMENTO-SALVAR
     else
      -
  end-if
. retorna
```

PROCEDIMENTO-ACRESCENTAR-ESQ-M

```
*/ usuário  tem  capacidade  de  mudar  função antes de cada
   término do enquanto /*
. enquanto  função.tipo   seja   definição  o  sistema  lê
  elemento.classe
  if elemento.classe = entidade
     then
      . chama PROC-ACRESCENTAR-ENTIDADE
     else
     if elemento.classe = relacionamento
        then
         . chama PROC-ACRESCENTAR-RELACIONAMENTO
        else
        if elemento.classe = associação
           then
            . chama PROC-ACRESCENTAR-ASSOCIAÇÃO
           else
           if elemento.classe = agrupamento
              then
               . chama PROC-ACRESCENTAR-AGRUPAMENTO
              else
              if elemento.classe = elemento-agrupamento
                 then
                  . chama    PROC-ACRESCENTAR-ELEMENTO-
                    AGRUPAMENTO
                 else
                 if elemento.classe = esquema
                    then
                     . chama  PROC-ACRESCENTAR-ESQUEMA
                    else
                      -
                 end-if
               end-if
            end-if
         end-if
      end-if
   end-if
  end-enquanto
. retorna
```

PROC-ACRESCENTAR-ENTIDADE

. chama DEFINE-ENTIDADE
. coloca na tabela 4.2.6 ELEMENTOS-ACRESCENTADOS
  entidade.nome, entidade.classe
. retorna

PROC-ACRESCENTAR-RELACIONAMENTO

. chama DEFINE-RELACIONAMENTO
. coloca na tabela 4.2.6 ELEMENTOS-ACRESCENTADOS relacionamento.nome, relacionamento.classe
. retorna

PROC-ACRESCENTAR-ASSOCIAÇÃO

. chama DEFINE-ASSOCIAÇÃO
. coloca na tabela 4.2.6 ELEMENTOS-ACRESCENTADOS
  associação.nome, associação.classe
. retorna

PROC-ACRESCENTAR-AGRUPAMENTO

. chama DEFINE-AGRUPAMENTO
. coloca na tabela 4.2.6 ELEMENTOS-ACRESCENTADOS
  agrupamento.nome, agrupamento.classe
. retorna

PROC-ACRESCENTAR-ELEMENTO-AGRUPAMENTO

. chama DEFINE-ELEMENTO-AGRUPAMENTO
. coloca na tabela 4.2.7 ELEMENTOS-ACRESC-DE-AGRUPAMENTO agrupamento.nome, agrupamento.elemento.nome e agrupamento.elemento.classe
. retorna

PROC-ACRESCENTAR-ESQUEMA

. chama DEFINE-ESQUEMA
. coloca na tabela 4.2.6 ELEMENTOS-ACRESCENTADOS
  esquema.nome, esquema.classe
. retorna

```
PROCEDIMENTO-REMOVER-ESQ-M


*/ usuário   tem  capacidade  de  mudar função antes de cada
   término do enquanto /*
. enquanto   função.tipo  seja    remoção  o   sistema    lê
  elemento.classe a remover
  if elemento.classe = entidade
     then
      . chama  PROC-REMOVER-ENTIDADE
      else
      if elemento.classe = relacionamento
         then
          . chama  PROC-REMOVER-RELACIONAMENTO
         else
         if elemento.classe = associação
            then
             . chama  PROC-REMOVER-ASSOCIAÇÃO
            else
            if elemento.classe = agrupamento
               then
                . chama PROC-REMOVER-AGRUPAMENTO
               else
               if elemento.classe = elemento.agrupamento
                  then
                   . chama  PROC-REMOVER-ELEMENTO-AGRUP
                  else
                  if elemento.classe = esquema
                     then
                      . chama  PROC-REMOVER-ESQUEMA
                     else
                      -
                  end-if
                end-if
             end-if
          end-if
       end-if
  end-enquanto
. retorna
```

PROC-REMOVER-ENTIDADE

. chama REMOVE-ENTIDADE
. coloca na tabela 4.2.8 ELEMENTOS-REMOVIDOS
  entidade.nome, entidade.classe
. retorna

PROC-REMOVER-RELACIONAMENTO

. chama REMOVE-RELACIONAMENTO
. coloca na tabela 4.2.8 ELEMENTOS-REMOVIDOS
  relacionamento.nome, relacionamento.classe
. retorna

PROC-REMOVER-ASSOCIAÇÃO

. chama REMOVE-ASSOCIAÇÃO
. coloca na tabela 4.2.8 ELEMENTOS-REMOVIDOS
  associação.nome, associação.classe
. retorna

PROC-REMOVER-AGRUPAMENTO

. chama REMOVE-ENTIDADE
. coloca na tabela 4.2.8 ELEMENTOS-REMOVIDOS
  agrupamento.nome, agrupamento.classe
. retorna

PROC-REMOVER-ELEMENTO-AGRUP

. chama REMOVE-ELEMENTO-AGRUPAMENTO
. coloca na tabela 4.2.9 ELEMENTOS-REMOVIDOS-DO-AGRUP.
  agrupamento.nome, agrupamento.elemento.nome,
  agrupamento.elemento.classe
. retorna

PROC-REMOVER-ESQUEMA

. chama REMOVE-ENTIDADE
. coloca na tabela 4.2.8 ELEMENTOS-REMOVIDOS
  esquema.nome, esquema.classe
. retorna

```
PROCEDIMENTO-ALTERAR-ESQ-M


*/ usuário   tem  capacidade  de  mudar função antes de cada
   término do enquanto /*
. enquanto    função.tipo   seja   alteração   o   sistema lê
  elemento.classe a alterar
 . aguardar o usuário indicar elemento.classe a alterar
   if elemento.classe = entidade
      then
       . chama  PROC-ALTERAR-ENTIDADE
      else
      if elemento.classe = relacionamento
         then
          . chama  PROC-ALTERAR-RELACIONAMENTO
         else
         if elemento.classe = associação
            then
             . chama  PROC-ALTERAR-ASSOCIAÇÃO
            else
            if elemento.classe = agrupamento
               then
                . chama  PROC-ALTERAR-AGRUPAMENTO
               else
                -
             end-if
          end-if
       end-if
    end-if
end-enquanto
. retorna
```

PROC-ALTERAR-ENTIDADE

. chama ALTERA-ENTIDADE
. coloca na tabela 4.2.10 ELEMENTOS-ALTERADOS
  antiga.entidade.nome, nova.entidade.nome
. retorna

PROC-ALTERAR-RELACIONAMENTO

. chama ALTERA-RELACIONAMENTO
. coloca na tabela 4.2.10 ELEMENTOS-ALTERADOS
  antigo.relacionamento.nome, novo.relacionamento.nome
. retorna

PROC-ALTERAR-ASSOCIAÇÃO

. chama ALTERA-ASSOCIAÇÃO
. coloca na tabela 4.2.10 ELEMENTOS-ALTERADOS
  antiga.associação.nome, nova.associação.nome
. retorna

PROC-ALTERAR-AGRUPAMENTO

. chama ALTERA-AGRUPAMENTO
. coloca na tabela 4.2.10 ELEMENTOS-ALTERADOS
  antigo.agrupamento.nome, novo.agrupamento.nome
. retorna

MODO-ATRIBUTO-M

```
. usuário aponta função desejada (área 4)
. enquanto função desejada não for salvar ou fin faça
  if função.tipo = definição
     then
      . chama  PROC-ACRESCENTA-ATRIBUTO-ESQ-EXI
      else
      if função.tipo = remoção
         then
          . chama  PROC-REMOVE-ATRIBUTO-ESQ-EXI
         else
         if função.tipo = alteração
            then
             . chama  PROC-ALTERA-ATRIBUTO-ESQ-EXI
             . chama  PROC-ALTERA-DOMÍNIO-ESQ-EXI
             else
             if função.tipo = listar
                then
                 . chama  PROCEDIMENTO-LISTAR
                 else
                  -
              end-if
           end-if
        end-if
     end-if
. end-enquanto
  if função.tipo = salvar
     then
      . chama  PROCEDIMENTO-SALVAR
     else
      -
  end-if
. retorna
```

PROC-ACRESCENTA-ATRIBUTO-ESQ-EXI

. usuário aponta elementos a definir atributos
. torna invisíves elementos não apontados
. chama PROC-DEFINE-ATRIBUTO
. coloca na tabela 4.2.11 ATRIBUTOS-ACRESCENTADOS
  atributo.nome, atributo.domínio
. retorna

PROC-REMOVE-ATRIBUTO-ESQ-EXI

. usuário aponta elementos a remover atributos
. torna invisíves elemento nào apontados
. chama PROC-REMOVE-ATRIBUTO
. coloca na tabela 4.2.12 ATRIBUTOS-REMOVIDOS
  atributo.nome, atributo.domínio
. retorna

PROC-ALTERA-ATRIBUTO-ESQ-EXI

. usuário aponta elementos a alterar atributos
. torna invisíves elementos não apontados
. chama PROC-ALTERA-ATRIBUTO
. coloca na tabela 4.2.13 ATRIBUTOS-ALTERADOS atributo.nome
. retorna

PROC-ALTERA-DOMÍNIO-ESQ-EXI

. redesenha conteúdo área 3
    . TEXTO(ATRIBUTOS)
. usuário aponta domínio
. chama PROC-ALTERA-DOMÍNIO
. coloca na tabela 4.2.14 DOMÍNIOS-ALTERADOS domínio.tipo
. retorna

PROCEDIMENTO-SALVAR

. enviar ao GERPAC informação referente a:
    . esquema e versão
    . conteúdo das tabelas correspondentes
. retorna

PROCEDIMENTO-LISTAR

. imprimir na impressora informação contida na área 1
. retorna

```
PROC-MANIPULAR-DADOS

. chama PROC-TELA-MANIPULA-DADOS
. chama PROC-LER-ESQUEMA
. lê entrada
  if entrada = consultar
     then
      . chama PROC-CONSULTAR
     else
     if entrada = inserir
        then
         . chama PROC-INSERIR
        else
        if entrada = alterar
           then
            . chama PROC-ALTERAR
           else
           if entrada = remover
              then
               . chama PROC-REMOVER
              else
               -
           end-if
         end-if
      end-if
   end-if
. retorna
```

PROC-TELA-MANIPULA-DADOS

```
. desenha área 1
. desenha área 2
. desenha área 3
. desenha área 4
. desenha conteúdo área 2
      . desenha =
      . desenha ≠
      . desenha <
      . desenha >
      . desenha ≤
      . desenha ≥
      . TEXTO( NOT )
      . TEXTO( OR  )
      . TEXTO( FIM-CONSULTA )
. desenha conteúdo área 3
      . TEXTO(1.CONSULTAR)
      . TEXTO(2.INSERIR)
      . TEXTO(3.REMOVER)
      . TEXTO(4.ALTERAR)
      . TEXTO(FIM)
. desenha conteúdo área 4
      . TEXTO(APONTE FUNÇÃO)
      . TEXTO(APONTE PRIMITIVA)
. retorna
```

PROC-LER-ESQUEMA

. lê dados do esquema e versão fornecidos
. preenche tabelas ENTIDADE, RELACIONAMENTO, ASSOCIAÇÃO e AGRUPAMENTO
. retorna

PROC-CONSULTAR

```
 loop enquanto consultar (apontamento na área 4)
   loop enquanto não fim-consulta (apontamento na área 2)
   . entrada dados
   case
     objeto:(área1)
                        . realça objeto
                        . mostra atributos
                        . armazena p/consulta
                        . break;
     atributo:(área1)
                        . realça atributo
                        . armazena p/consulta
                        . break;
     condição:(área2)
                        . pede atributo1/valor
                        . pede atributo2/valor
                        . armazena cláusula p/ consulta
                        . break;
     fim-condição:
                        . faz consistência da claúsula completa
                        . guarda claúsula
                        . inicia  outra cláusula
     fim-consulta:
                        . break;

   end-case
   end-loop não fim-consulta
    . faz consistência
    . chama  GERPAC passando "código consulta" e "cláusula
      (s) de consulta"
    . aguarda resposta
    . solicita forma de amostragem da informação
    .  if amostragem = impressora
          then
          . imprime
          else
          . faz a amostragem no vídeo na forma de janela
    . pergunta se deseja seguir consultando
end-loop consultar
. retorna
```

```
PROC-INSERIR


. loop enquanto inserir
  . loop enquanto não fim-inserção
  . entrada dados
    case
     objeto:
                    . realça objeto
                    . mostra atributos
                    . armazena
                    . break;
     atributo:
                    . realça atributo
                    . entrada do valor a inserir
                    . armazena informação
                    . break;
     fim-inserção:
                    . break;
   end-case
 end-loop
 . faz consistência
 . chama GERPAC passando "código inserção" e "valor de
   inserção"
 . aguarda resposta
 . consulta se deseja seguir inserindo
.end-loop
. retorna
```

PROC-ALTERAR

```
. loop enquanto alterar
  . loop enquanto não fim-alteração
  . entrada dados
    case
     objeto:
                    . realça objeto
                    . mostra atributos
                    . armazena
                    . break;
     atributo:
                    . realça atributo a alterar
                    . pede valor
                    . armazena informação
                    . break;
     condição:
                    . pede atributo1/valor
                    . pede atributo2/valor
                    . armazena informação
                    . break;
     ou       :
                    . faz consistência cláusula completa
                    . guarda cláusula
                    . inicia p/ outra parte
     fim-alteração:
                    . break;
 end-case
 end-loop
 . faz consistência
 . chama GERPAC passando "código alteração" e "claúsula
   de alteração"
 . aguarda resposta
 . consulta se deseja seguir alterando
end loop
. retorna
```

```
PROC-REMOVER


. loop enquanto remove
. entrada de dado
  objeto:
                . realça objeto
                . mostra atributos
                . armazena
* monta claúsula
. loop enquanto não fim-remoção
. entrada dados
  case
     objeto:
                    . realça objeto
                    . mostra atributos
                    . armazena
                    . break;
     atributo:
                    . realça atributo
                    . mostra informação
                    . armazena
                    . break;
     condição:
                    . pede atributi1/valor
                    . pede atributo2/valor
                    . armazena informação
                    . break;
     ou       :
                    . faz consistência de claúsula completa
                    . guarda claúsula
                    . inicia p/ outra parte
     fim-remoção:
                    . break;
  end-case
  end-loop
. faz consistência
. chama GERPAC passando "código remoção" e "cláusula remoção"
. aguarda resposta
. consulta se deseja seguir removendo
end-loop
. retorna
```

```
PROC-HELP


. desenha contorno área total
. desenha conteúdo área total
      . TEXTO (ESTA É UMA INTERFACE GRÁFICA)
      . TEXTO (INTERATIVA DE COMUNICAÇÃO)
      . TEXTO (USUÁRIO-BANCO DE DADOS.)

      . TEXTO (PARA A OPERAÇÃO DO SISTEMA)
      . TEXTO (TEM-SE TRÊS MODOS :)

      . TEXTO (- CRIAR -)
      . TEXTO (- MANIPULAR ESQUEMA-)
      . TEXTO (- MANIPULAR DADOS -)



      . TEXTO ( MODO CRIAÇÃO )
      . TEXTO ( **** *******

      . TEXTO ( PERMITE DEFINIR A BASE DE DADOS )

      . TEXTO ( MODO MANIPULAÇÃO DE ESQUEMA)
      . TEXTO ( **************************)
      . TEXTO ( PERMITE OPERAR A BASE DE DADOS )
      . TEXTO ( A NÍVEL DE ESQUEMA.)

      . TEXTO ( MODO MANIPULAÇÃO DE DADOS )
      . TEXTO ( ************************* )
      . TEXTO ( PERMITE OPERAR OS DADOS DA BASE)
      . TEXTO ( DE DADOS)
. retorna
```