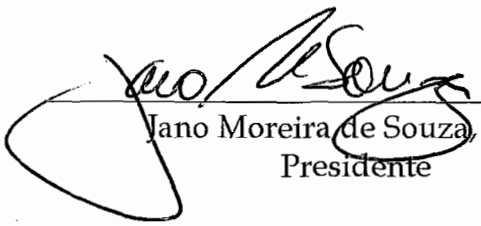
GEOCOPPE
UM SGBD ESPACIAL PARA APLICAÇÕES GEOGRÁFICAS

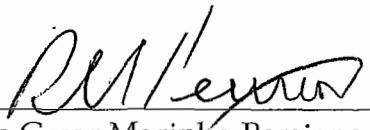
Patricia Auler

Tese submetida ao Corpo Docente da Coordenação dos Programas de Pós-graduação de Engenharia da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do grau de Mestre em Ciências em Engenharia de Sistemas e Computação

Aprovado por:



Jano Moreira de Souza, Ph.D.
Presidente



Ronaldo Cesar Marinho Persiano, D.Sc.

Ubiratan Porto dos Santos, Ph.D.

Rio de Janeiro, RJ - Brasil
Abril de 1993

AULER, PATRICIA
Geocoppe: Um SGBD Espacial para Aplicações Geográficas 1993
XII, 120 p. 29,7 cm (COPPE/UFRJ, M.Sc., Engenharia de Sistemas e Computação, 1993)
Tese - Universidade Federal do Rio de Janeiro, COPPE
1. SGBD espacial
I. COPPE/UFRJ
II. Geocoppe: Um SGBD Espacial para Aplicações Geográficas 1993

AGRADECIMENTOS

Ao Instituto de Pesquisas da Marinha e ao CNPq pelo apoio financeiro dado através de bolsas de estudos.

Ao professor Jano Moreira de Souza pelo dedicado trabalho de orientação.

Ao Programa de Engenharia Oceânica por incentivar que seus funcionários se especializem, permitindo que eu obtivesse o meu mestrado concomitante às minhas atividades de analista de sistemas do Programa.

A Diretoria de Hidrografia e Navegação pelo fornecimento dos dados apresentados nos exemplos desta tese.

A Marta Lima Queiróz Mattoso pelo grande auxílio e pelas pacientes explicações relativas ao COPPEREL.

A Milton Ramirez pela atenção e minuciosa revisão desta tese.

A Paulo Cesar Colonna Rosman pelo renovado incentivo e auxílio na revisão da tese.

Aos meus pais e amigos pelo incondicional apoio.

Resumo da tese apresentada à COPPE/UFRJ como parte do requisitos necessários para a obtenção do grau de Mestre em Ciências

GEOCOPPE
UM SGBD ESPACIAL PARA APLICAÇÕES GEOGRÁFICAS

Patricia Auler

Abril de 1993

Orientador: Jano Moreira de Souza
Programa: Engenharia de Sistemas e Computação

Os Sistemas de Informação Geográfica (SIG) possuem quatro componentes principais. Um destes componentes é um sistema de armazenamento e recuperação dos dados.

Nesta tese foi desenvolvido um Sistema de Gerenciamento de Bases de Dados (SGBD) espacial GEOCOPPE, que além de armazenar e recuperar dados realiza também funções de sobreposição de mapas e cálculo de propriedades espaciais. O sistema GEOCOPPE foi implementado sobre o SGBD relacional COPPEREL, desenvolvido pelo Programa de Engenharia de Sistemas e Computação. Para o GEOCOPPE lidar com os dados espaciais foram desenvolvidos três novos tipos de dados: ponto, linha e região. Estes tipos foram implementados através da estrutura de dados *quadtree*.

O GEOCOPE é formado por três componentes: o COPPEREL, o Gerente de Dados Espaciais (GDE) e o Processador de Consultas Exibição (PCE). O GDE armazena no COPPEREL as *quadtrees*, realizando operações sobre as mesmas. O PCE realiza a interface do usuário com o sistema, e une os dados espaciais aos dados não espaciais.

Considerando que o COPPEREL foi escrito em FORTRAN para ambiente de micro-computadores em sistema DOS, a implementação do GEOCOPPE seria extremamente difícil devido às limitações de memória e à pouca versatilidade da linguagem FORTRAN para tratamento gráfico. Em vista disso, antes do desenvolvimento do GEOCOPPE, foi necessário traduzir o código do COPPEREL para linguagem C, de modo a transportá-lo para o ambiente de estação gráfica SUN sob o sistema operacional UNIX. Posteriormente foi acrescentado ao COPPEREL o tipo de dados inteiro sem sinal.

Para exemplificar os comandos implementados do GEOCOPPE são utilizadas informações oceanográficas de salinidade e temperatura da água do mar, importantes para a Acústica Submarina.

Abstract of Thesis Presented to COPPE/UFRJ in Partial Fulfillment of the Requirements for the Degree of Master of Sience

GEOCOPPE
A SPATIAL DBMS FOR GEOGRAPHIC APPLICATIONS

Patricia Auler

April de 1993

Chairman: Jano Moreira de Souza
Major Department: Engenharia de Sistemas e Computação

The Geographic Information Systems (GIS) have four main components. One of these components is a system for storing and retrieving data.

In this thesis it has been developed a spatial Data Base Management System (DBMS) called GEOCOPPE, that in addition to store and retrieve data, performs superposition of maps and calculations with spatial properties. The GEOCOPPE system was developed for the relational DBMS called COPPEREL, that belongs to the Programa de Engenharia de Sistemas e Computação. For GEOCOPPE to deal with spatial data, three different types of data have been developed: point, line, and region. Those types were implemented using the data structure of *quadtrees.*

The GEOCOPPE system comprises three components: the COPPEREL, the Spatial Data Manager (GDE), and the Consults and Exhibitions Processor (PCE). The SDM stores the *quadtrees* in the COPPEREL and performs operations on them. The CEP is the user interface with the system, and joints the spatial and non-spatial data.

Since COPPEREL has been written in FORTRAN language, for DOS environment in micro-computers, the implementation of GEOCOPPE would be extremely difficult due to memory limitations and to the lack of versatility of FORTRAN to deal with graphic features. For that reason, prior to GEOCOPPE development, COPPEREL has been translated to C language, and transported to the UNIX environment in a SUN workstation. In addition, the signless integer data type was included in COPPEREL .

To exemplify the commands implemented in GEOCOPPE, data concerning salinity and temperature of sea water have been used; such data are important to Submarine Acoustics.

Sumário

Lista de Figuras

Lista de Tabelas

1. Introdução

O objetivo desta tese é o desenvolvimento de um sistema de gerenciamento de dados espaciais, voltado para informações geográficas, GEOCOPPE, acoplado a um sistema de banco de dados relacional previamente desenvolvido, COPPEREL. O GEOCOPPE foi desenvolvido como um primeiro passo na direção de um sistema para automatizar manipulações e análise de dados espaciais. O sistema foi concebido como uma interface superior ao Sistema de Gerenciamento de Base de Dados COPPEREL, existente no Programa de Engenharia de Sistema e Computação da COPPE/UFRJ, e que para tal teve que ser significativamente ampliado e modificado.

O GEOCOPPE possui três tipos de dados espaciais: ponto, linha e região. Estes três tipos básicos são suficientes para manipular todos os dados existentes nos mapas. Estes tipos foram implementados utilizando a estrutura de dados hierárquica *quadtree*, pois esta estrutura apresenta excelente desempenho em operações de sobreposição, que são muito utilizadas em aplicações geográficas, além de economizar bastante espaço de armazenamento.

As pessoas lidam diariamente com informações espaciais, pois vivem num mundo que é basicamente espacial. Constantemente, gerenciam e analisam com facilidade informações que envolvam distância, adjacência além de relações espaciais mais complexas. As manipulações espaciais que realizamos, apesar de sua complexidade, são feitas automática e intuitivamente. Entretanto, tais manipulações revelam-se muito difíceis de serem automatizadas através de algoritmos.

Partindo da premissa de criar uma interface para um sistema previamente desenvolvido, a primeira etapa no desenvolvimento desta tese, foi a transformação de todo o código do sistema COPPEREL, escrito em linguagem de programação FORTRAN, para a linguagem de programação C. Tal tarefa foi realizada em estações de trabalho SUN sob sistema operacional UNIX, com auxílio de um conversor de linguagem.

Foi implementado no COPPEREL o tipo de dados inteiro sem sinal, pois este tipo era necessário para a implentação dos tipos espaciais.

Este trabalho está dividido em quatro capítulos. O capítulo 1 é esta introdução. O capítulo 2 apresenta os conceitos básicos existentes nos Sistemas de Informação Geográficas (SIG). Posteriormente, são apresentadas as principais classificações de SIGs encontradas hoje em dia na literatura, e discute-se os principais subsistemas de um SIG. Como exemplo de SIGs são abordados dois sistemas muito difundidos atualmente: o ARC/INFO e o SPANS. O ARC/INFO utiliza para

representar seus tipos espaciais o modelo topológico, enquanto o SPANS utiliza a *quadtree* e um modelo vetorial (topológico). Através da utilização da *quadtree* o SPANS afirma que consegue uma economia de espaço no disco da ordem de 3 a 8 vezes, além de agilizar as operações de sobreposição de mapas.

O capítulo 3 apresenta o GEOCOPPE como uma proposta para um Sistema de Gerenciamento de Base de Dados Espacial. É descrita sua arquitetura, seus componentes principais, e funções. A estrutura de dados adotada para representar os dados espaciais no GEOCOPPE, a *quadtree*, e seus algoritmos são descritos. Por último, os algoritmos principais e a interface utilizada é apresentada.

O capítulo 4 descreve as relações utilizadas para exemplificar diversos comandos do GEOCOPPE.

No capítulo 5 apresentam-se as conclusões e recomendações para continuar o desenvolvimento do sistema GEOCOPPE, objeto desta tese. Finalmente, no anexo A, exemplifica-se, uma seção típica de consulta ao sistema desenvolvido. O exemplo apresentado é baseado em dados muito utilizados na área de Acústica Submarina. O dados foram obtidos na Diretoria de Hidrografia e Navegação (DHN).

2. Sistemas de Informação Geográfica (SIG)

Os SIGs surgiram devido a necessidade crescente por sistemas computacionais para aplicações que envolvam informações com características espaciais ou geográficas [Marbl90][Dange90]. Tais aplicações ocorrem, por exemplo, nas áreas de acústica submarina (velocidade do som na água, temperatura e salinidade do mar), gerenciamento de recursos naturais (agricultura, pecuária e ecologia), gerenciamento urbano (rede de telefonia e de energia, tráfico viário), gerenciamento costeiro (batimetria, perfis de praia), gerenciamento hídrico (construção de barragens) dentre muitas outras [Burro86] [Marbl90].

Os SIGs são poderosos conjuntos de ferramentas para o agrupamento, armazenamento, recuperação, transformação e apresentação dos dados espaciais retirados do mundo real, ao se focar um objeto particular [Burro86]. Estes dados espaciais são encontrados normalmente sob a forma de mapas analógicos e cada um tem o atributo especial de possuir uma localização única na Terra. Os mapas tem sido, historicamente, a principal estrutura usada na maioria das disciplinas, que envolvem análise espacial. Eles fornecem um meio para o armazenamento, concepção de idéias, análise de conceitos, predição de cenários futuros, desenvolvimento de decisões sobre a geografia, e a comunicação de idéias para outras pessoas [Dange90].

A análise e recuperação dos mapas normalmente envolve observação visual e análise intuitiva que são realizados com o auxílio de ferramentas simples (ex. escalas e planímetros). Observa-se que enquanto é fácil recuperar pequenas quantidades de informações, é muito trabalhoso, além de consumir bastante tempo, a recuperação de grandes quantidades de informações ou a determinação quantitativa de relacionamentos complexos que podem existir entre os elementos dos mapas [Marbl90]. Além disto, as alterações são muito caras, pois os mapas muitas vezes tem que ser totalmente refeitos e não existe nenhum mecanismo que assegure que as mudanças realizadas a um mapa sejam transportadas aos mapas associados.

Muitas das técnicas desenvolvidas em outras ciências, tais como o planejamento urbano e gerenciamento de recursos naturais, se tornam limitadas na prática, sem a capacidade e o processamento muito rápido oferecido pelos SIGs ao se lidar com o grande volume de observações requerido por essas aplicações. Com isto, vemos que os SIGs se tornaram cada vez mais uma necessidade para várias aplicações.

O primeiro esforço para a aplicação de computadores para reduzir os problemas obtidos com a manipulação manual dos mapas estava relacionado com

aplicações militares. Mas somente a partir da década de 60 é que os primeiros SIGs foram desenvolvidos, normalmente financiados pelo governos. Um dos primeiros a ser desenvolvido foi o *Canada Geographic Information System*, que continha informações sobre agricultura, floresta, vida animal, capacidades de recreações, etc [Anten91]. Com o passar do tempo, os computadores foram se tornando bem mais baratos e acessíveis, e a demanda por informações espaciais foi rapidamente crescendo. Hoje, os SIGs são utilizados não somente pelos governos mas também por diversas companhias privadas.

2.1. Tecnologias relevantes

A tecnologia da informação geográfica foi desenvolvida através de diversos esforços paralelos, mas independentes, que se estendem por várias disciplinas. Conforme estas disciplinas e suas tecnologias se desenvolvem, seus avanços são incorporados à tecnologia dos SIGs [Anten91]. Portanto, os SIGs são sistemas grandes e multidisciplinares. Entre as principais disciplinas envolvidas temos:

- Banco de Dados;
- Computação Gráfica;
- Geometria Computacional;
- Processamento de Imagens;
- Inteligência Artificial;
- Sensoriamento Remoto;
- Geografia;
- Cartografia;
- Fotogrametria.

2.2. Classificação dos SIGs

Ainda hoje existe muita incerteza quanto ao significado da palavra SIG, isto é, o que pode ser considerado como um SIG. Isto tem feito com que se chame qualquer sistema que apresente um mapa na tela de SIG [Cowen90]. Por exemplo, segundo Antennuci [Anten91] os SIGs não são necessariamente sistemas computacionais, apesar da maioria deles estar aumentando o grau de automação. Já Clarke afirma que os SIGs são quaisquer sistemas auxiliados por computadores para a captação, o armazenamento, a recuperação, a análise e a apresentação dos dados espaciais [Clark86].

São encontradas na literatura, basicamente, quatro maneiras para se definir o que é um SIG [Cowen90]. A primeira, chamada de metodologia orientada a processo, é baseada na idéia de que um sistema de informação consiste de vários subsistemas integrados que ajudam a converter o dado geográfico em informação útil. Todo o sistema tem que incluir rotinas para entrada, armazenamento, recuperação, análise e saída das informações geográficas. Estes sistemas são avaliados segundo a sua capacidade de fornecer informações úteis em tempo hábil. O problema desta abordagem é que a mesma não ajuda a distinguir os SIGs de, por exemplo, sistemas de cartografia. A segunda maneira é a abordagem da aplicação, segundo a qual os SIGs são categorizados de acordo com o tipo de informação que está sendo manipulada. Desta forma tem-se sistemas de planejamento urbano, sistemas de gerenciamento dos recursos naturais, etc [Pavli82]. Nesta abordagem, o problema reside em não se conseguir distinguir os SIGs de outras formas de processamento de dados geográficos. A terceira maneira, a abordagem orientada a ferramentas, advém da idéia que os SIGs possuem um conjunto sofisticado de rotinas e algoritmos para a manipulação dos dados espaciais. Desta forma são enumerados subsistemas [Marbl90] [Burro86] ou funções [Dange90] que um SIG deve possuir. Esta abordagem é muito útil para avaliar sistemas diferentes, mas, segundo Cowen [Cowen90] não fornece uma definição viável. A quarta maneira é a abordagem de banco de dados. Esta considera que um SIG é melhor definido como um sistema que usa um banco de dados espacial e provê respostas a perguntas de natureza geográfica. Deste modo um SIG genérico pode ser visto como um conjunto de rotinas espaciais especializadas que atuam sobre um Sistema de Gerenciamento de Banco de Dados (SGBD).

Cowen acrescenta a estas quatro maneiras mais uma: segundo ele, um SIG é um sistema capaz de responder a seguinte pergunta: "Que regiões cobrem uma determinada área ?". Ressaltando que esta pergunta pode ser aplicada também

a pontos e linhas. Isto, porque dentre todas as operações que são comumente atribuídas aos SIGs, as únicas que são somente encontradas nestes são: a busca espacial e a sobreposição de mapas. Para responder a pergunta citada acima é necessário que se apliquem estas duas operações.

Nesta tese é adotada a abordagem orientada por ferramentas apresentada por Marble em [Marbl90]. Segundo tal abordagem um sistema só é considerado como um SIG se ele possuir quatro subsistemas (apresentados a seguir), sendo que os quatro subsistemas tem que processar os dados eficientemente. Deste modo, a simples adição de poucas e ineficientes funções de manipulação de dados espaciais a um sistema que é orientado apenas a uma das funções básicas não o torna um SIG verdadeiro.

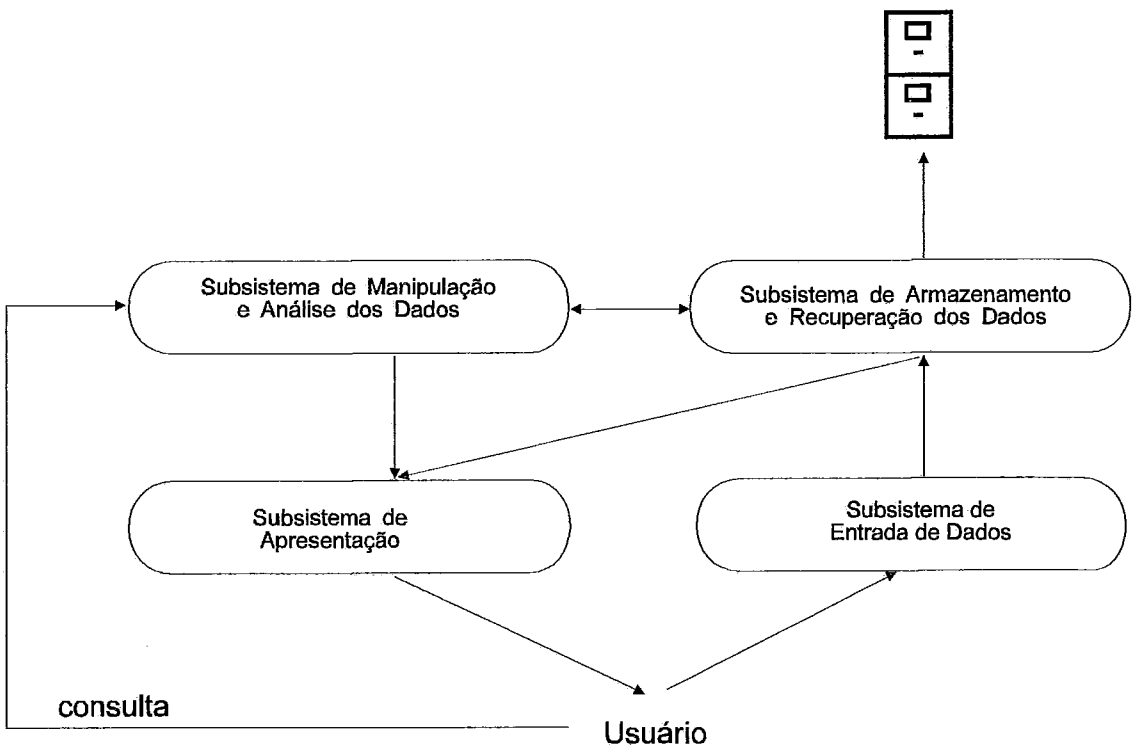


Figura 1 - Principais Componentes de Software

2.2.1. Subsistema de entrada dos dados

Este subsistema é responsável pela coleta e o processamento dos dados espaciais que são obtidos através de mapas, documentos feitos a mão, fotografias aéreas, fotografias de satélites, dados de campos, etc. Existem três formas básicas de dados espaciais: pontos, linhas e regiões [Burro86] [Samet90a] [Marbl90]. A estas formas associamos três tipos de informações: as

alfanuméricas (nome do elemento, cor, etc), as geométricas (coordenadas, comprimentos e ângulos) e as relações de vizinhança (quais elementos são vizinhos a um dado elemento). Portanto, é necessário que se associe os dados espaciais aos seus atributos não espaciais, ligando-os de algum modo.

Existem diversos métodos para dar entrada aos dados espaciais. A escolha do melhor método é muito influenciada pela aplicação, pelo orçamento disponível e pelo tipo de dado a ser entrado [Burro86]. Os dados podem ser fornecidos, por exemplo, pelo teclado, por uma mesa digitalizadora, ou por um *scanner*.

A entrada de dados é provavelmente, no presente, o maior "gargalo" nos sistemas de gerenciamento de dados espaciais, e representa o maior custo individual na maioria dos projetos, especialmente quando a base de dados é muito grande. Futuramente, este custo deve diminuir com o desenvolvimento de ferramentas digitais, pois os dados já serão capturados na forma digital, evitando assim a transformação dos dados da forma analógica para a forma digital.

Durante a entrada dos dados podem ser introduzidos erros, tanto nos dados espaciais quanto nos dados não espaciais. Os erros são agrupados do seguinte modo [Burro86]:

- dados espaciais incompletos ou duplicados;
- dados espaciais no lugar errado;
- dados espaciais numa escala errada;
- dados espaciais distorcidos;
- dados espaciais ligados a dados não espaciais errados;
- dados não espaciais incompletos ou com valores errados.

Uma das maneiras de se saber se os dados espaciais, que existiam anteriormente sob a forma de mapas, foram introduzidos corretamente ou não no computador é fazer com que o SIG imprima-os, numa folha que seja de preferência transparente, na mesma escala do mapa original. Então, compara-se sistematicamente o mapa original com o gerado pelo computador, de modo que se detecte e se corrija os erros [Burro86].

Muitas vezes, é mais difícil corrigir os atributos dos dados espaciais pois eles podem estar incorretos, mas possuírem valores aceitáveis.

2.2.2. Subsistema de apresentação dos dados

Este subsistema é capaz de apresentar ao usuário, sob forma de tabelas ou mapas, parte ou todo o banco de dados original, bem como dados resultantes de manipulações e dados resultantes de modelagens espaciais. A apresentação envolve a chamada cartografia computadorizada ou digital.

A maioria dos usuários dos SIGs querem que os dados dos sistemas sejam apresentados de modo similar aos mapas convencionais. Desta forma os mapas produzidos pelos SIGs devem possuir informações tais como: legenda ou classes de intervalo, um título, um indicador de orientação e escala.

2.2.3. Subsistema de manipulação e análise dos dados

Este subsistema realiza uma variedade de tarefas tais como alterar a forma dos dados através de regras de agregações definidas pelo usuário, ou a produção de estimativas de parâmetros e restrições para vários modelos de simulações e otimizações espaço-temporais.

Entre os métodos de análise disponíveis temos alguns que são encontrados em todos os SIGs, como o cálculo de área e perímetro de polígonos. Outros mais específicos não são encontrados em todos os SIGs, como por exemplo, a análise digital de terrenos envolvendo desenho 3-D, cálculo de seções transversais e horizontais, cálculo de linhas de contorno, etc. Os métodos de análise espacial são usados com o objetivo de responder às perguntas submetidas pelo usuário.

2.2.4. Subsistema de armazenamento e recuperação dos dados

Este subsistema organiza os dados espaciais numa forma que permita a eles serem rapidamente recuperados pelo usuário para subseqüente análise. Ele também tem que permitir que as atualizações e correções na base de dados espacial sejam realizadas rapidamente.

2.3. Consultas

Existem vários tipos de consultas que podem ser feitas sobre os dados espaciais e às informações não espaciais associadas a eles. Algumas destas consultas envolvem análise espacial enquanto outras não. A seguir são apresentados alguns tipos de consultas fornecidas pelos SIGs atuais.

1) **Consulta por janela** - O sistema permite ao usuário gerar polígonos irregulares, quadrados ou círculos que funcionarão como uma janela. Esta janela será usada para realizar a recuperação espacial de pontos, linhas e regiões (polígonos), podendo ser feita num dos seguintes modos:
   a) por análise de adjacência, por exemplo recuperar um ponto que esteja mais próximo ao ponto especificado (janela).
   b) recuperação dos pontos, linhas e regiões que estejam contidos na janela especificada.
   c) recuperação dos pontos, linhas e regiões que passem pela janela especificada, isto é, não necessitam estar totalmente contidos nela.

2) **Resumo estatístico e recuperação por atributos booleanos** - O usuário especifica condições sobre as informações não espaciais dos objetos geométricos, e o sistema recupera os objetos que as satisfazem. Se for pedido, ele poderá também apresentar um resumo estatístico. Por exemplo o usuário poderia fazer a seguinte consulta "Forneça todas as regiões de um certo tipo de solo que possuem área maior que 50 m^2 e produza também um resumo estatístico dos perímetros, áreas e o total".

Para se realizar o primeiro tipo de consulta apresentado acima os SIGs fornecem duas operações espaciais importantes:

1) **Sobreposição dos dados espaciais** - Consiste em se criar um outro mapa que será o resultado da sobreposição de dois outros mapas que já existiam na base de dados. A sobreposição pode ser de polígonos ou áreas, neste caso poderá ser feita a união ou a interseção dos polígonos. Existe também a sobreposição de pontos com polígonos (ou áreas). O resultado desta operação consiste no conjunto de pontos mas com um atributo a mais, que indica a qual polígono o ponto pertence. Finalmente, tem-se a sobreposição de linhas com polígonos (ou áreas). O resultado desta operação consiste no conjunto

de linhas mas com um atributo a mais, que indica a qual polígono a linha, ou parte dela pertence.

2) **Geração de contorno externo (*buffer*)** - Consiste em se criar um polígono em torno de um dado espacial. A distância com que o polígono será criado do dado é variável e determinada pelo usuário. Esta operação é necessária para determinar a proximidade espacial de feições geográficas. Por exemplo, quando se quer saber que casas estão a menos de 10 metros da praia, cria-se um mapa de contorno externo dos mapas de linhas que definem as praias, e sobrepõe-se o mapa resultante com o mapa de ocupação urbana. A figura 2 apresenta exemplos de contornos externos para os 3 tipos de dados espaciais.

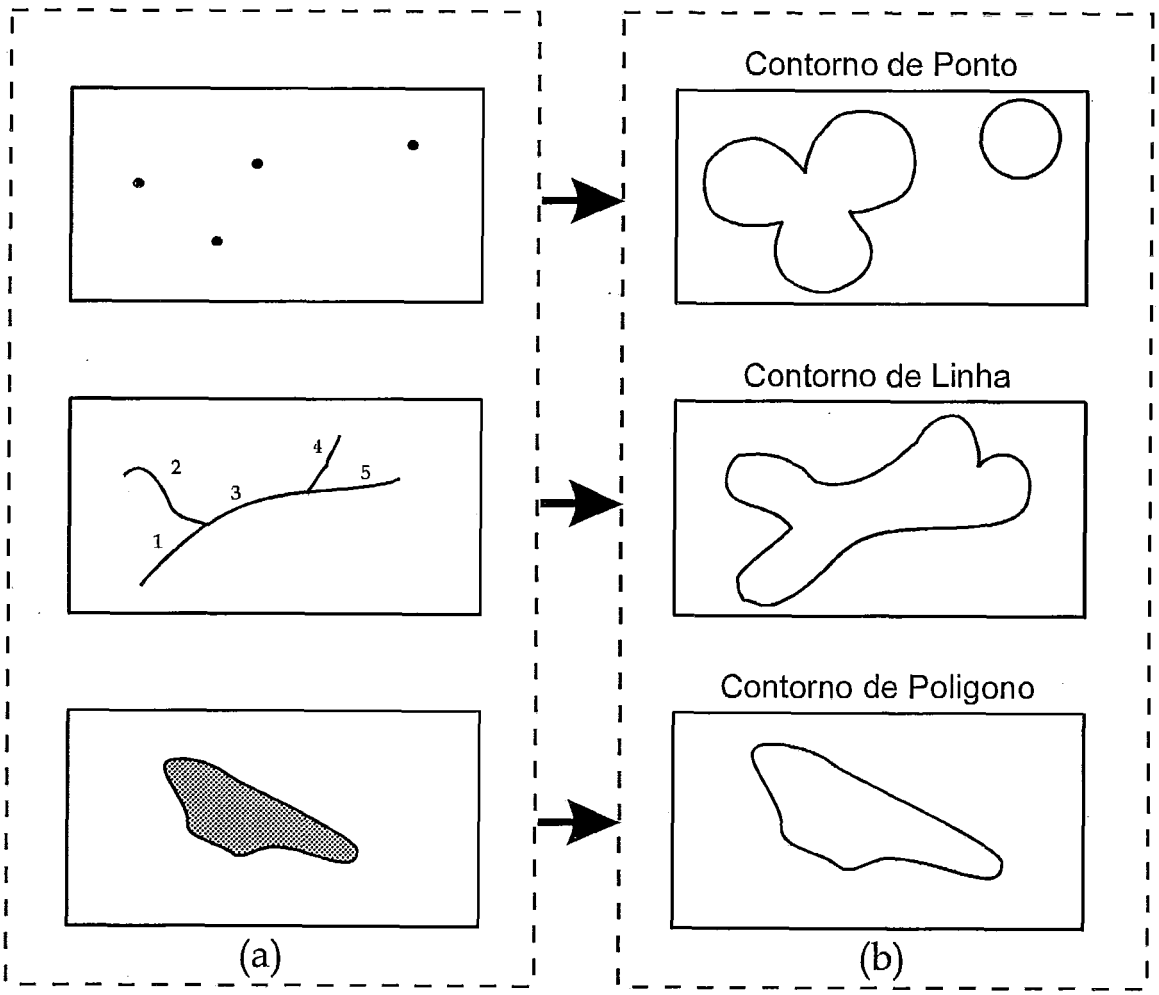


Figura 2 - Geração de contorno externo

Obviamente, outra operação muito importante existente nos SIGs é a busca dos dados espaciais dentro da base de dados.

2.4. Exemplos de SIGs

A seguir serão apresentados brevemente dois dos SIGs mais importantes de modo a dar uma visão geral das funcionalidades oferecidas pelos produtos atuais.

2.4.1. ARC/INFO

O sistema ARC/INFO foi desenvolvido pela ESRI de Redlands, California [Esri]. Ele utiliza um modelo de dados híbrido, pois usa um SGBD (Sistema de Gerenciamento de Banco de Dados) relacional INFO, desenvolvido pela Henco Corporation, para armazenar os dados não espaciais, e um módulo especial (ARC), que utiliza um modelo de dados topológico para armazenar e realizar operações sobre os dados espaciais [ARC/INFO]. Portanto, os dados espaciais são gerenciados pelo módulo ARC, que é o programa principal do ambiente ARC/INFO. Este contém comandos que chamam outros subsistemas, e possui também funções para manipulação dos dados espaciais. Versões mais recentes permitem a utilização de SGBD's relacionais de uso geral, tais como ORACLE, INGRES no lugar do SGBD INFO.

O ARC/INFO suporta três tipo de dados espaciais: ponto, linha (arco) e polígono. A cada um desses tipos podem ser associados atributos. Portanto, o sistema admite que sejam fornecidas coordenadas apenas pertinentes a um desses três tipos de dados.

As informações espaciais são descritas através de suas coordenadas e relacionamentos topológicos. As coordenadas são utilizadas para localizar os dados espaciais no espaço, enquanto os dados topológicos são necessários para identificar arcos, nós e relacionamentos entre polígonos.

A criação dos arquivos de topologia (arquivos com a extensão ARC) é feita automaticamente. O sistema ARC/INFO associa aos segmentos de linha (arcos), que são descritos por pares de coordenadas, os identificadores dos polígonos que estão a direita e a esquerda, e os seus dois nós de conexão.

As informações não espaciais armazenadas pelo módulo INFO são unidas às informações espaciais através de várias tabelas que permitem o relacionamento entre os dois tipos de informações. Estas tabelas são mantidas atualizadas pelo sistema ARC/INFO.

ARC/INFO aceita vários formatos padrões de entrada e saída, entre eles temos:

- USGS DLG-3 formatos padrão e opcional;

- arquivo GBF/DIME ;
- Imagem ERDAS e arquivos GIS
- Intergraph Standard Interchange Format (SIF)
- arquivos de projeto do AutoCAD

ARC/INFO possui uma linguagem de comandos com a qual é possível se fazer consultas sobre os dados, e a construção de macros. Através desta linguagem é possível se realizar funções analíticas, tabulares e gráficas. Cada comando possui um nome em linguagem natural que descreve a sua função. Por exemplo, o comando DIGITIZE chama o sistema de digitalização de arcos.

Acoplando a linguagem de comandos do ARC/INFO com a AML (*Arc Macro Language*), que possui macros e ferramentas para construção de menus, é possível para o usuário criar aplicações específicas.

A AML é uma linguagem de programação de quarta geração, projetada para ser independente do *hardware*, que facilita o uso de variáveis, realiza quebra de fluxo lógico e *loops*, manipula cadeias de caracteres e de textos, realiza operações aritméticas e trigonométricas, faz chamadas e passa variáveis para outros programas escritos em AML, e realiza operações do SIG selecionadas. A AML também fornece facilidades para o usuário criar seus próprios menus.

O ARC/INFO foi projetado de forma modular. O sistema é formado por nove módulos, incluindo os módulos ARC e INFO. Sendo que os três últimos módulos são vendidos separadamente e fornecem funções mais especializadas.

1) ARC - É o responsável pela ativação de todos os outros módulos, tendo condição de realizar as seguintes funções:
   - Conversão de dados para entrada e saída;
   - Edição e digitalização de mapas;
   - Fornecer comandos para verificação e correção de erros;
   - Funções para a transformação e para a projeção de coordenadas que são úteis para a sobreposição e a união de mapas adjacentes;
   - Gerenciamento e manipulação dos atributos dos dados espaciais;
   - Operações analíticas que incluem a criação de contornos externos, sobreposição de mapas, análise do vizinho mais próximo e relatórios resumidos das estatísticas;

   Muitas das funções de ARC são comandos que podem ser executados de maneira não interativa, sem a necessidade de dispositivos gráficos, ou de interação com o usuário.
2) INFO - Como foi dito anteriormente, o módulo INFO é um SGBD relacional completo, usado para armazenar as informações não espaciais do sistema ARC/INFO. As informações não espaciais são unidas às espaciais através de

quatro arquivos INFO: um arquivo TIC (escala), um arquivo BND (*Boundary*), um arquivo PAT (*Polygon Attribute Table or Point Attribute Table*), e um arquivo AAT (*Arc Attribute Table*). Estes arquivos são acessados pelo módulo INFO, para consulta, análise, atualizações e relatórios. Através de comandos do ARC/INFO pode-se incluir colunas nos dois últimos tipos de arquivos.

Dentro do módulo INFO o usuário tem acesso as informações espaciais através de identificadores armazenados nas tabelas acima (mantidas atualizadas pelo ARC/INFO) . Acoplado aos identificadores estão algumas propriedades dos dados espaciais (ex. área e perímetro).

3) ADS (***Arc Digitizing System***) - Este módulo é utilizado para digitalização e edição de dados de linhas, áreas e pontos. Durante a digitalização, não é necessária nenhuma rotina especial para a definição dos relacionamentos espaciais, tais como a definição dos identificadores de polígonos que estão a direita/esquerda, dos polígonos que estão dentro de outros, etc. Isto será realizados após a digitalização através de comandos do ARC/INFO (ex. BUILD e CLEAN).
4) ARCEDIT - Este módulo é um editor de base de dados e de gráficos. Ele combina a capacidade de edição das funções existentes nos sistemas CAD, com o poder das bases de dados geográficas. No ARCEDIT pode-se copiar, acrescentar, remover, mudar de formato, e atualizar pontos, linhas, áreas e anotações de mapas. Cada vértice individual dentro de uma linha pode ser movido, acrescentado ou removido. Pode-se mudar o formato das linhas, suavizá-las (*spline*) ou ainda alinhá-las. O ARCEDIT também permite diversas formas de edição sobre as anotações, pode-se gira-las, escala-las, espaça-las proporcionalmente, etc.
5) ARCPLOT - O ARCPLOT é um módulo de cartografia iterativa e mapeamento. Através deste sistema é possível a geração de uma grande variedade de mapas em diversos dispositivos tais como: plotters de pena e eletrostáticos, impressoras gráficas (laser, jatos de tinta, matricial), ou na própria tela do computador.
6) LIBRARIAN - Este módulo é o responsável pelo gerenciamento de bancos de dados cartográficos muito grandes. O LIBRARIAN divide a base de dados em pequenos pedaços chamados de *tiles*. A função de inserção de mapas automaticamente divide e indexa cada pedaço do mapa de acordo com o *tile* definido pelo usuário.
7) NETWORK - Este módulo permite duas categorias de funções: endereçamento de geocódigos e análise de redes. Através do NETWORK é possível achar, por exemplo, o caminho ótimo numa rede ou análise de tempo/distância.

8) TIN (*Triangulated Irregular Network*) - Este módulo é um conjunto de programas utilizados para o armazenamento, gerenciamento e a análise de superfícies tridimensionais. O TIN representa uma rede irregular de triangularização, que é um conjunto de triângulos não sobrepostos e adjacentes utilizados para representar as faces de uma superfície. TIN fornece funções para a conversão de dados, modelagem e apresentação de superfícies. Entre as funções de modelagens tem-se o cálculo de volume, aspecto e comprimento de superfícies, geração de polígonos *voronoi*, etc.
9) COGO - COGO é o módulo de coordenadas geométricas. Este módulo é utilizado para suportar as funções realizadas, por exemplo, por engenheiros civis no projeto e layout de subdivisões e ruas.

2.4.2. SPANS (*SPatial ANalysis System*)

O SIG SPANS foi desenvolvido pela TYDAC Technologies Corp. of Arlington com a colaboração da IBM. O SPANS integra, analisa e modela dados espaciais. SPANS foi desenvolvido em C, e roda em microcomputadores tipo PC usando aproximadamente 2.5 Mbytes de disco rígido.

O SPANS pode ler, escrever e analizar tanto o dado vetorial quanto o dado *raster*. Os dados vetorias são usados para representar redes e outras entidades lineares, e como um meio para trocar informações geográficas entre diversas bases de dados digitais. Com o objetivo de trocar dados com outros sistemas, o SPANS lê os seguintes formatos de troca de dados:

- DLG;
- ARC-EXPORT;
- MOSS;
- GRASS;
- Intergraph (SIF);
- AutoCad (SXF);
- GIMS.

No modo *raster*, o SPANS possui uma interface que acomoda diversos arquivos *raster*, entre eles:

- Modelo de Elevação Digital (DEM);
- ERDAS;
- PCI;
- DIPIX;
- Meridian;
- I2S.

Os dados não espaciais podem ser importados e exportados em diversos formatos ASCII.

O SPANS utiliza a estrutura de dados *quadtree* para armazenar e sobrepor (*overlay*) níveis temáticos. A Tydac afirma que o SPANS, com a utilização da *quadtree*, se torna extremamente eficiente, e reduz de 3 a 8 vezes o espaço de armazenamento sobre muitos formatos *raster* e vetoriais. Fazendo com que o sistema possa acomodar banco de dados *raster* muito grandes (tão grande quanto 32.768 x 32.768).

O SPANS, tal qual o ARC/INFO, foi projetado de forma modular, sendo constituído por sete módulos, que são unidos através de um menu integrado. Todas as funções podem ser acessadas através de menus. Cada um dos módulos é descrito a seguir:

1) CORE - Este módulo possui os componentes básicos do SPANS. Ele inclui funções para a análise, manipulação e apresentação dos mapas; funções para realizar operações de sobreposição de mapas; operações de proximidade; funções para modelagem; etc. Embora muitas das operações utilizem as *quadtrees*, CORE também utiliza estrutura de dados *raster* e vetoriais para apresentar e analisar os dados. CORE pode, entre outras potencialidades, medir distâncias lineares, produzir contornos externos para pontos, linhas e polígonos e definir polígonos Voronoi, que identificam a zona de influência para um conjunto de pontos.
2) TYDIG - Este módulo realiza funções de digitalização e edição dos mapas. Através do TYDIG e de uma tabela de digitalização, é possivel se capturar pontos, linhas e polígonos de mapas em papel. O sistema contém diversas capacidades de edição que permitem a correção de erros feitos durante a digitalização. Durante a digitalização podem ser associados atributos as entidades.
3) POTMAP - Este módulo cria superfícies a partir de um conjunto de pontos, o que permite que observações pontuais sejam transformadas em mapas temáticos.
4) CONTOUR - Este módulo contrói mapas temáticos a partir de dados pontuais, usando o método de interpolação da Malha de Triangularização Irregular (*Triangulated Irregular Network - TIN* -), com método linear e não linear.
5) RASTER - Usado para importar dados no formato *raster*. Por exemplo, para importar imagens de satélites, imagens de fotografias aéreas "escaniadas".
6) VECTOR - Usado para importar dados em formatos vetoriais.
7) MAP INDEXING - Este módulo fornece um método eficiente para o armazenamento, atualização e disseminação de informações detalhadas das conversões dos mapas por níveis temáticos.

2.5. Sistema de Gerenciamento de Banco de Dados (SGBD) Espacial

Pela definição dos componentes de um SIG apresentada na seção 2.2 vimos que o subsistema de armazenamento e recuperação de dados, que representa o SGBD espacial, não é responsável pela parte de consulta a base de dados espacial. Isto porque, pela definição apresentada, o SGBD espacial não possui as funções de análise espacial necessárias a realização das consultas espaciais. Os SGBDs convencionais não são capazes de manipular eficientemente grandes volumes de dados espaciais. Isto tem sido o principal obstáculo para o desenvolvimento de base de dados globais [Marbl90].

Algo parecido aconteceu com o desenvolvimento dos SGBD convencionais. No princípio, os dados eram armazenados em arquivos e desenvolviam-se para cada aplicação, ou conjunto de arquivos, rotinas de acesso aos dados. Caso a forma de armazenamento dos dados fosse alterada os programas necessitariam ser mudados. Com o tempo, os programas passaram a fazer chamadas a um módulo externo, e este se encarregava do armazenamento dos dados. Assim, incorporando cada vez mais funções, foram surgindo os SGBDs, tais quais eles são hoje em dia.

Pode-se dizer que um SGBD espacial é o responsável pelo armazenamento, pela recuperação e pela manipulação dos dados espaciais. Observe que a operação de manipulação não esta incluída na definição apresentada na seção 2.2. Mas, do mesmo modo que os SGBDs convencionais incorporaram funções, se tornando o que são hoje em dia, os SGBDs espaciais passam a incorporar algumas funções de manipulação para os dados espaciais. Deste modo, estes além de possuirem todas as características encontradas nos SGBDs convencionais (consultas não procedimentais, recuperação de falhas, mecanismos de segurança, compartilhamento dos dados, concorrência, etc), permitem a manipulação dos dados espaciais. Um SGBD espacial é o módulo central do SIG com o qual todos os outros módulos trocam informações, realizando consultas, inserindo e obtendo os dados necessários às suas operações.

Hoje, existe uma predominância na utilização de SGBDs convencionais para o armazenamento e manipulação dos atributos não espaciais que estão associados às informações espaciais, junto com a utilização de programas especializados para o armazenamento, recuperação e a manipulação dos dados espaciais. O SIG ARC/INFO é um bom exemplo desta abordagem, onde o SGBD relacional INFO é utilizado para manipular os atributos não espaciais, enquanto o sistema ARC manipula os atributos espaciais [Marbl90].

Algumas das mais importantes pesquisas em SIG estão agora se concentrando no projeto de SGBDs espaciais ótimos que liguem as informações geográficas aos atributos ou variáveis associadas as entidades geográficas que estão sendo representadas pelo sistema [Cowen90]. Dentro desse esforço, concorda-se que a criação de um SIG tem que começar com a escolha de um modelo de dados adequado [Peuqu90]. Segundo Pequet, o grande responsável pelo fracasso de muitos SIGs no passado foi a má escolha do modelo de dados.

Não existe até o momento nenhuma teoria matemática coerente das relações espaciais, isto é, não existe nenhum conjunto claramente definido de relacionamento espaciais entre as entidades geográficas [Marbl90] [Tomli90]. Além disto, não existe identificada nenhuma categorização de consultas espaciais que possa ser especificada em termos das operações que elas requerem que sejam realizadas sobre os dados espaciais [Tomli90]. Isto é sentido na dificuldade em se desenvolver um SGBD verdadeiramente espacial e na criação de algoritmos eficientes para a manipulação de dados espaciais.

Grande parte dos SGBDs espaciais existentes possui dificuldades na incorporação de novos tipos de dados e de adaptação a uma aplicação que seja diferente daquela para a qual ele foi desenvolvido. Tal acontece porque existe uma grande variedade de tipos de dados espaciais, e aplicações que os utilizam, fazendo com que normalmente se desenvolvam SGBDs voltados para uma determinada aplicação.

O problema da falta de versatilidade e dificuldade de integração é somado ao fato dos SGBDs espaciais atuais estarem encontrando problemas muito sérios com o grande volume de dados, e com o tempo necessário para o processamento destes (eficiência) [Peuqu90].

2.5.1. Modelo de dados espaciais

Nenhum modelo ou abstração da realidade pode representar todos os seus aspectos. É impossível projetar um modelo de dados que seja igualmente útil em todas as situações. Por exemplo, temos modelos que são eficientes para a análise espacial e não o são para a plotagem, e vice-versa.

Ao se escolher um modelo de dados para um SGBD espacial deve-se ter em mente quais as operações que serão mais utilizadas, o volume de dados e a eficiência desejada.

Um modelo de dados mais completo, que represente explicitamente mais entidades e relacionamentos espaciais, será mais complexo, acarretando maior

dificuldade de compreensão, manutenção e dificultando a otimização do armazenamento e consultas, e provavelmente implicando em maior gasto em armazenamento. Por outro lado, um modelo de dados desenvolvido de forma que represente apenas os elementos necessários para uma tarefa particular, excluindo os relacionamentos e entidades que não são relevantes, será mais simples e provavelmente implicará num menor gasto em armazenamento, porém dará menor flexibilidade e capacidade de expansão à base de dados.

Como foi dito anteriormente, as três formas básicas de tipos de dados espaciais são ponto, linha e região. Para representá-las digitalmente existem diversos modelos de dados espaciais que podem ser classificados em dois grupos: modelos vetoriais e modelos de *grid* [Peque90], vide figura 3. O segundo grupo surgiu com a computação.

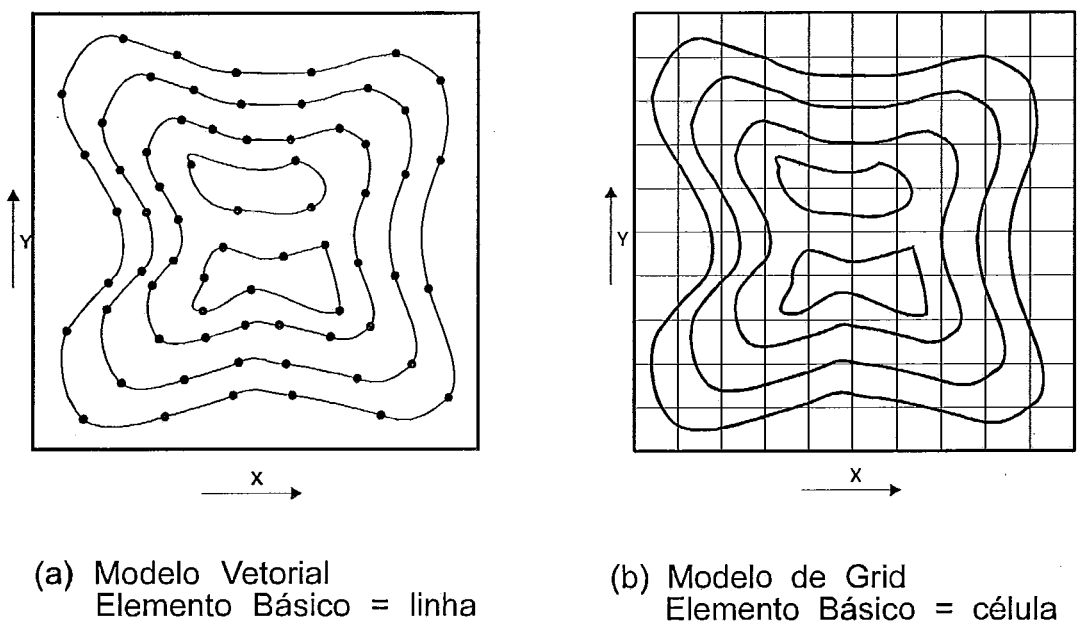


Figura 3 - Modelo Vetorial e *Grid*

O modelos vetoriais evoluiram a partir da cartografia, e representam o mundo real através de um conjunto de linhas e pontos que representam a localização e/ou o limite das entidades. Os modelos vetoriais armazenam uma seqüência de pontos para representar os dados espaciais. Uma série de coordenadas (x, y) de pontos são modelados como componentes de um única linha. Os pontos podem ser representados nesta organização como segmentos de comprimento zero. Fazem parte deste grupo os seguintes modelos: modelo Spaghetti, modelo Topológico, GBF/DIME, POLYVRT, *chain codes*.

Os modelos *grid* surgiram junto com a computação, e caracterizam-se pela divisão do espaço em vários pedaços. Cada pedaço armazena informações sobre o(s) objeto(s) espacial(is) que estão dentro dele. Fazem parte deste grupo os seguintes modelos: tri-cell [Falou89], arquivo de *grid* [Hutfl88] [Neive84] [Salzb86] e *quadtrees* [Samet83b] [Samet84b].

Dependendo do modelo *grid* adotado irão variar: o número máximo de subdivisões, o momento em que elas ocorrem, o número máximo de objetos que podem estar contidos numa célula (pedaço) e se as células obtidas com uma divisão são do mesmo tamanho ou não.

Os modelos de dados do tipo *grid* representam o dual lógico dos modelos de dados vetoriais. Nos modelos vetoriais, as entidades individuais passam a ser as unidades básicas de dados para o qual a informação espacial é explicitamente armazenada. Enquanto nos modelos *grid* a unidade básica é a unidade do espaço para o qual a informação da entidade é explicitamente armazenada [Peuqu90].

Nos modelos vetoriais os relacionamentos espaciais (direita, esquerda, perto de, etc) tem que ser explicitamente armazenados. Enquanto que nos modelos de *grid* estes relacionamentos fazem parte do modelo, isto é, são inerentes ao modelo.

Não existe um modelo ótimo. Cada um se adapta melhor a determinadas operações, fazendo com que a escolha de um modelo de dados para um SGBD espacial seja dependente da aplicação e dos dados a serem representados.

3. Proposta de um SGBD espacial

3.1. Arquitetura geral

O sistema GEOCOPPE pode ser classificado como sendo do grupo de SGBD espacial baseado no acoplamento de um SGBD convencional e um módulo para o gerenciamento dos dados espaciais. Neste caso específico utilizamos o SGBD relacional COPPEREL [Matto87] [Zakim85] [Gonça90] e um Gerente de Dados Espaciais (GDE) baseado em *quadtrees*. As *quadtrees* serão apresentadas no item 3.3.2.

O GEOCOPPE possui algumas funções de análise espacial, portanto, ele é mais do que simplesmente um subsistema de armazenamento e recuperação de dados. No entanto ele não pode ser classificado como um SIG pois não possui, ou não realiza eficientemente, algumas das funções apresentadas no capítulo 2.

O GDE pode ser encarado como uma camada superior ao COPPEREL. Ele resolve todas as operações que envolvam dados espaciais (inclusão, consulta, remoção, etc). O GDE utiliza o COPPEREL, via rotinas de nível mais baixo, para armazenar e acessar as *quadtrees* como se elas fossem relações, e, também, utiliza o COPPEREL através da interface embutida para armazenar as outras relações (apresentadas no item 3.3) usadas por ele, [Trott89]. O GDE é o responsável pelo gerenciamento das *quadtrees*, não sabendo, por exemplo, que as *quadtrees* de região por ele gerenciadas são utilizadas para representar atributos do tipo região. Esta arquitetura foi escolhida visando o aproveitamento de tudo que foi desenvolvido a nível de controles operacionais para o COPPEREL, centrando os esforços de desenvolvimento nos módulos relativos aos dados espaciais.

Portanto, o sistema proposto adiciona muito ao que já existe no COPPEREL. Este pode ser alterado sem a necessidade de preocupações maiores com o GEOCOPPE, tais como no caso da inclusão do mecanismo para a reconstrução após falha [Gonça90], desenvolvido no início dos trabalhos no GEOCOPPE.

Entretanto, não poderíamos utilizar a linguagem padrão do COPPEREL para a definição e manipulação de dados (LOPEREL [Zakim85]), pois esta não teria como interpretar as partes relativas aos dados espaciais. A solução adotada foi construir uma interface que separasse estas partes e enviasse aos módulos GDE e COPPEREL as respectivas requisições.

Devido à natureza da aplicação (informações gráficas), achou-se por bem construir tal interface "gráfica". Foi utilizado um sistema de janelas (Xview [Helle90]), menus *pop-up* acionados por botões gráficos via apontamento direto, que podem chamar *forms* (devido muito ao fato de estar-se trabalhando em es-

tações de trabalho SUN que já fornecem várias ferramentas para a construção de tal interface).

Portanto, esta interface processa as consultas do usuário e apresenta os resultados em janelas gráficas. Tal interface foi denominada Processador de Consultas e Exibição - PCE, e sua posição na arquitetura pode ser vista na figura 4.

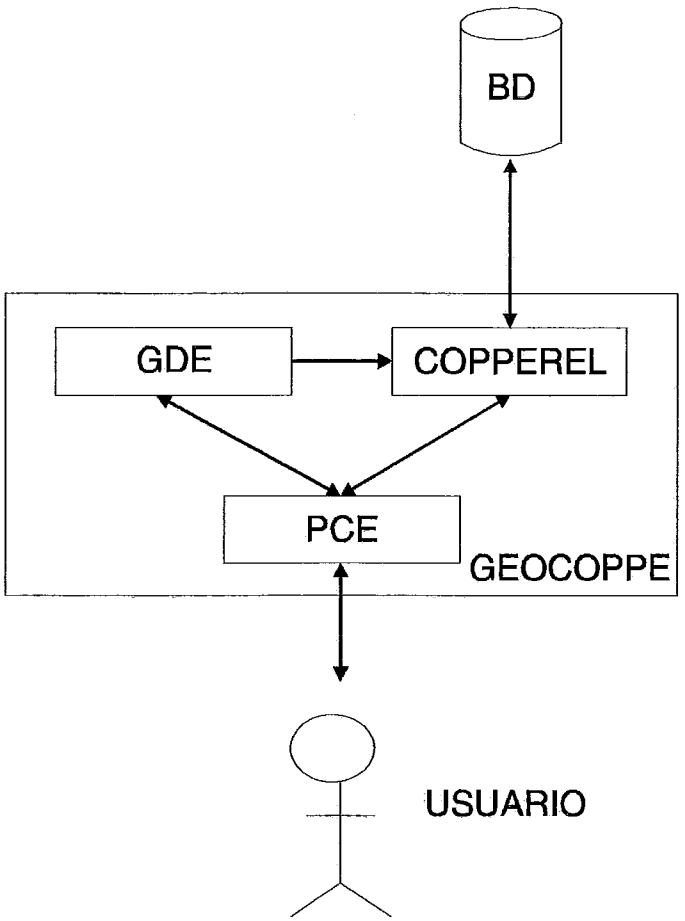


Figura 4 - Arquitetura do GEOCOPPE

O acoplamento entre o GDE e o COPPEREL é realizado pelo PCE, com o auxílio de duas novas relações: REL_ESPACIAIS e ATR_ESPACIAIS. Estas duas relações fazem parte do esquema espacial, e realizam o gerenciamento dos dados espaciais. A relação REL_ESPACIAIS armazena as relações que possuem algum atributo espacial, e a relação ATR_ESPACIAIS armazena os atributos espaciais dessas relações. Tanto a REL_ESPACIAIS quanto a ATR_ESPACIAIS não fazem parte do esquema do COPPEREL, sendo tratadas por este praticamente como relações comuns. A única diferença entre as relações do esquema espacial e as relações comuns do COPPEREL, é que as relações do esquema espacial são do tipo -1. Como o valor -1 para tipo não é utilizado pelo COPPEREL, empregou-se tal valor para caracterizar as relações do esquema espacial no GEOCOPPE. As relações REL_ESPACIAIS

e a ATR_ESPACIAIS inexistem tanto para o COPPEREL quanto para o GDE, só sendo manipuladas pelo PCE. O GDE gerencia as *quadtrees*, não "sabendo" como estas são utilizadas para armazenar os atributos espaciais do GEOCOPPE.

A janela principal do PCE, figura 5, possui cinco botões e uma sub-janela. Quando o usuário seleciona um botão, apresenta-se ao usuário um menu de comandos do COPPEREL. O primeiro botão (*Sessão*) é referente ao conceito de sessão do COPPEREL, com opções para abrir uma base de dados, iniciar uma sessão, etc. O segundo botão (*Gerência*) dá acesso ao menu de comandos de gerência da base de dados, tais como, criar relações, vistas, índices, etc. O terceiro botão abre o menu de (*Manipulação*), onde temos comandos de seleção, união, etc. Através do quarto botão (*Entrada\Saída*) acessamos os comandos de inserção, remoção de registros, etc. O último botão (*Copperel*) abre uma janela de texto para a entrada de comandos COPPEREL diretamente (janela Comandos Copperel). Este texto é enviado diretamente para o compilador de comandos COPPEREL e, portanto, não deve conter operações com dados espaciais. A figura 6 mostra os menus que estão associados aos botões de gerência, manipulação e entrada e saída.

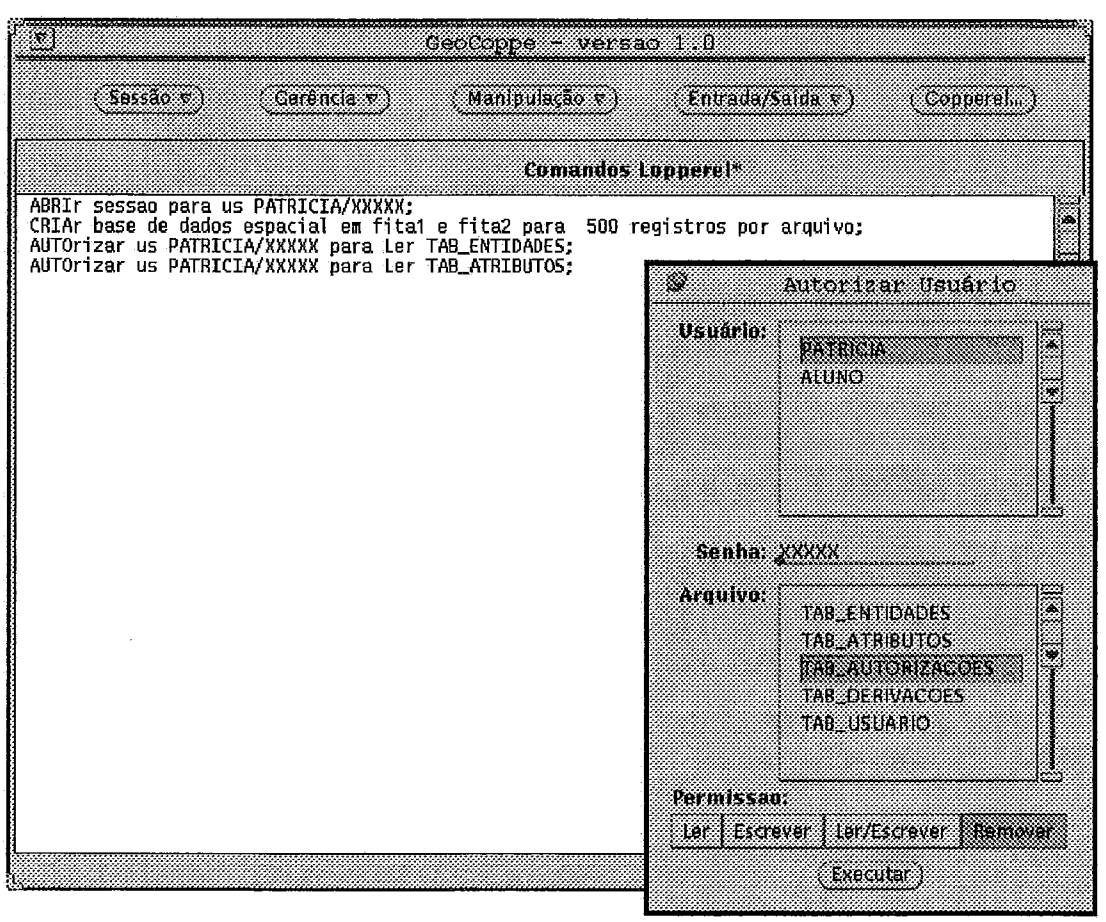


Figura 5 - Janela Principal do PCE, e a janela do comando autorizar usuario

Caso o usuário envie comandos sobre os dados espaciais diretamente, através da janela Comandos Copperel (figura 25), o COPPEREL não o realizará, enviando para o PCE uma mensagem de erro. Isto porque o COPPEREL foi alterado de forma a só realizar operações sobre as relações espaciais quando autorizado pelo PCE. Desta forma evita-se que o usuário possa tornar a base de dados espacial inconsistente. Por exemplo, a base tornar-se-ía inconsistente caso o usuário removesse uma relação espacial ou removesse uma tupla numa relação espacial, sem fazer com que o GDE atualizasse os atributos espaciais.

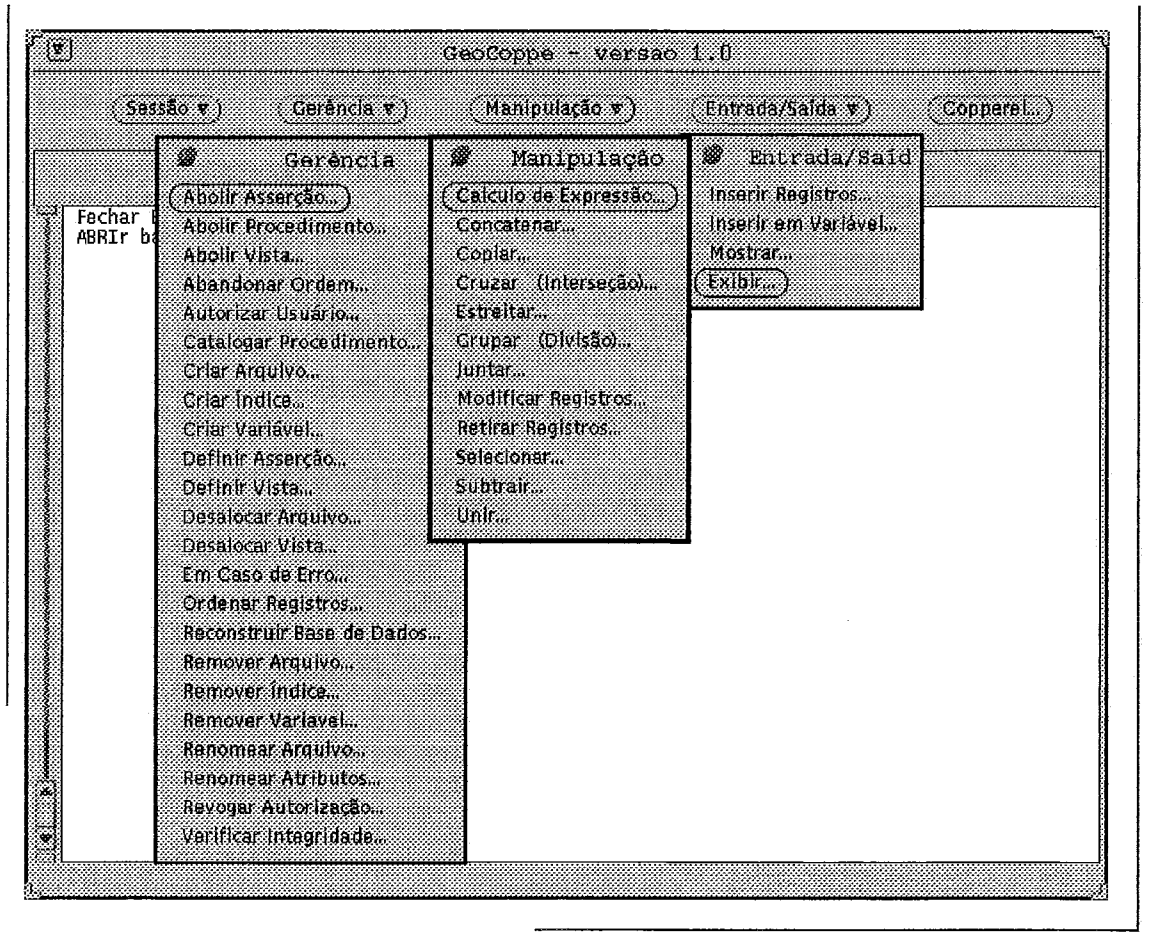


Figura 6 - Menus associados aos botões da janela principal do PCE

As consultas submetidas ao GEOCOPPE pela interface gráfica podem ser representadas por uma linguagem de consulta que foi denominada de LOPEREL*. A LOPEREL* é a Linguagem de Manipulação do Dados (LMD) e a Linguagem de Definição dos Dados (LDD) do GEOCOPPE, e é uma extensão da LOPEREL para manipular e definir os dados espaciais. Portanto, ela contém todos os comandos da LOPEREL e mais alguns, e os extende para manipular os dados espaciais.

A sub-janela da janela principal do PCE (figura 5) é utilizada para ecoar os comandos da LOPEREL* fornecidos pelo usuário através das outras janelas do PCE. Na figura 5, são mostrados quatro comandos que foram ecoados na sub-janela.

Neste sistema podemos realizar consultas não espaciais e espaciais, bem como combiná-las para se obter os resultados desejados, isto é, podemos ter seleção de atributo relacional com o resultado projetado sobre atributo espacial, e vice-versa.

3.2. Tipos de dados

O GEOCOPPE suporta além dos tipos de dados: inteiro, inteiro sem sinal, real e caracter; existentes no COPPEREL, os seguintes tipos espaciais: ponto, linha, região. O tipo inteiro sem sinal não existia no COPPEREL, sendo adicionado ao mesmo no início do desenvolvimento do GEOCOPPE. A localização dos dados espaciais é definida através do sistema cartesiano de coordenadas.

Quando se cria uma relação e especifica-se que um dos atributos será do tipo ponto, o GEOCOPPE pede que o usuário defina uma área no espaço, que conterá todos os pontos referentes ao atributo. Esta área é definida através da coordenada de um ponto e de uma distância, que funciona como largura e como altura do quadrado que irá conter todos os pontos da relação. O ponto fornecido corresponde a coordenada do canto superior esquerdo do quadrado. Esta região no espaço não é fixa, isto é, conforme são inseridas as tuplas da relação, a região aumenta, quando necessário, de forma que ela **sempre** contenha todos os pontos do atributo. No entanto este processo é bastante custoso, devendo ser evitado quando possível.

É necessário que se defina este quadrado também para os tipos linha e região, nestes casos, ele é definido para cada valor do atributo.

3.3. Gerenciador dos Dados Espaciais - GDE

Quando é submetida uma consulta espacial ao GEOCOPPE, o PCE não pode enviá-la diretamente ao COPPEREL, pois este não teria como resolvê-la, já que não reconhece os dados espacias (ponto, linha e região). Há portanto, a necessidade de preparar os dados para o PCE, o que é realizado pelo GDE. Através do GDE, os dados espaciais são gerenciados, armazenados no COPPEREL como relações, e efetuadas sobre eles as operações espaciais. Portanto, o PCE chama o GDE quando necessita operar sobre os dados espaciais.

O GDE manipula as seguintes relações do esquema espacial CODIGO_PONTO, CODIGO_SEGMENTO, SURROGATE, QUADTREE. Cada uma destas relações será apresentada posteriormente.

3.3.1. Catálogo de *quadtrees*

O GDE usa a *quadtree* como estrutura de armazenamento dos dados espaciais. Tal estrutura apresenta um bom desempenho nas operações de união e intercessão, em consultas por janela, entre outras operações.

Como mencionado anteriormente, através do GDE as *quadtrees* são armazenadas em relações do COPPEREL. O nome dessas relações é gerado através da combinação de um número único com a letra Q, que é colocada na frente do número. Para tanto basta converter o número para uma cadeia de caracter de tamanho 10 (suficiente para armazenar o maior inteiro sem sinal = 2^{32}), e acrescentar o caracter 'Q' na frente. Por exemplo, se o valor do número for 1234, ele é convertido para a cadeia "0000001234", e é acrescentado na sua frente o caracter 'Q', obtendo a cadeia de caracteres "Q0000001234", que representa o nome da relação que contém a *quadtree* desejada. Seria possível, ao invés de armazenar este número e ter que calcular o nome da *quadtree*, armazenar o próprio nome. No entanto, com isso, se gastaria mais espaço em disco, pois quando se armazena um inteiro sem sinal gasta-se 4 bytes, enquanto para armazenar o nome da *quadtree* como caracter gasta-se 12 bytes.

Portanto, seria possível para o sistema criar 2^{32} *quadtrees*, pois este é o número máximo obtido com o tipo inteiro sem sinal. No entanto, o número de *quadtrees* que podem ser criadas depende do número de relações criadas pelo usuário e do número máximo de relações permitido pelo COPPEREL. Pois, como mencionado, as *quadtrees* são armazenadas em relações. O número usado na formação do nome da relação que contém a *quadtree* é obtido somando 1 ao va-

lor contido na relação SURROGATE. Esta relação representa o número da última *quadtree* criada, e possui apenas uma única tupla.

RELAÇÃO SURROGATE			
Atributo	**Tipo**	**Chave**	**Descrição**
SUR	inteiro sem sinal	não	número do último surrogate criado
CÓDIGO	inteiro sem sinal	sim	existe apenas para evitar que o atributo SUR seja a chave da relação. O seu valor é sempre igual a 1.

Tabela 1 - Descrição da relação SURROGATE do GDE.

Futuramente, esta limitação poderá deixar de existir se as *quadtrees* deixarem de ser armazenadas como relações. As referências a elas não seriam feitas através do nome da relação que as representa, mas através do seu endereço físico na base de dados. Deste modo, a relação SURROGATE perderá a sua necessidade, deixando portanto de existir.

3.3.2. *Quadtrees*

O termo *quadtree* é usado para descrever uma classe de estruturas de dados hierárquicas [Samet82], cuja propriedade em comum é o fato delas se basearem na decomposição recursiva do espaço de interesse. Elas foram primeiramente desenvolvidas com o objetivo de economizar espaço em disco ao se agregar informações que possuam valor idêntico ou similar. No entanto, observa-se que o tempo de execução que se ganha com esta agregação em diversas operações é freqüentemente de igual ou maior importância [Samet90a].

A mais conhecida de todas, e a que deu origem a todos os outros tipos de *quadtrees*, é a *quadtree* de região. Para explicá-la deve-se considerar primeiramente uma matriz de elementos de figura (pixel), chamada de imagem, de dimensões $2^n \times 2^n$. Cada pixel pode ser branco ou preto (imagem binária). O pixel preto representa um pedaço interno da imagem enquanto o pixel branco representa um pedaço externo da imagem (fundo). Denomina-se bloco uma subregião quadrada da imagem, sendo que o maior bloco possível é a imagem toda. Enquanto os blocos não forem totalmente brancos ou totalmente pretos, a imagem inicial será subdividida em 4 blocos de mesmo tamanho. Portanto, obtem-se um árvore quaternária, onde o bloco que foi dividido é o bloco pai dos 4 blocos resultantes. Do mesmo modo os 4 blocos obtidos são filhos do bloco dividi-

do. Os blocos onde ocorreram divisões são também chamados de blocos cinzas, pois possuem blocos pretos e brancos no seu interior. Além disto, se o bloco pai está num nível *m* o bloco filho estará num nível *m-1*. Portanto, a imagem inteira (bloco raiz de tamanho $2^n \times 2^n$) está no nível *n*. Os blocos pretos e brancos serão as folhas da árvore formada.

Os quatro filhos de um bloco são nomeados de NO (noroeste), NE (nordeste), SO (sudoeste) e SE (sudeste) conforme mostra a figura 7.

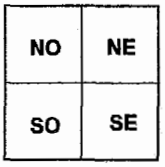


Figura 7 - Os quatro filhos obtidos com a divisão de um bloco

As *quadtrees* se diferenciam uma das outras pelo princípio de decomposição adotado, o tipo de dado que elas representam e a resolução (variável ou não). Por exemplo, a *quadtree* de região descrita acima, pode ser considerada uma estrutura de dados que possui resolução variável, e seu princípio de decomposição é o de um bloco ser dividido até se tornar todo branco ou todo preto.

A decomposição pode resultar em filhos do mesmo tamanho (decomposição regular) ou de tamanhos diferentes (decomposição irregular). Sendo esta última obtida quando a decomposição é governada pelos dados de entrada.

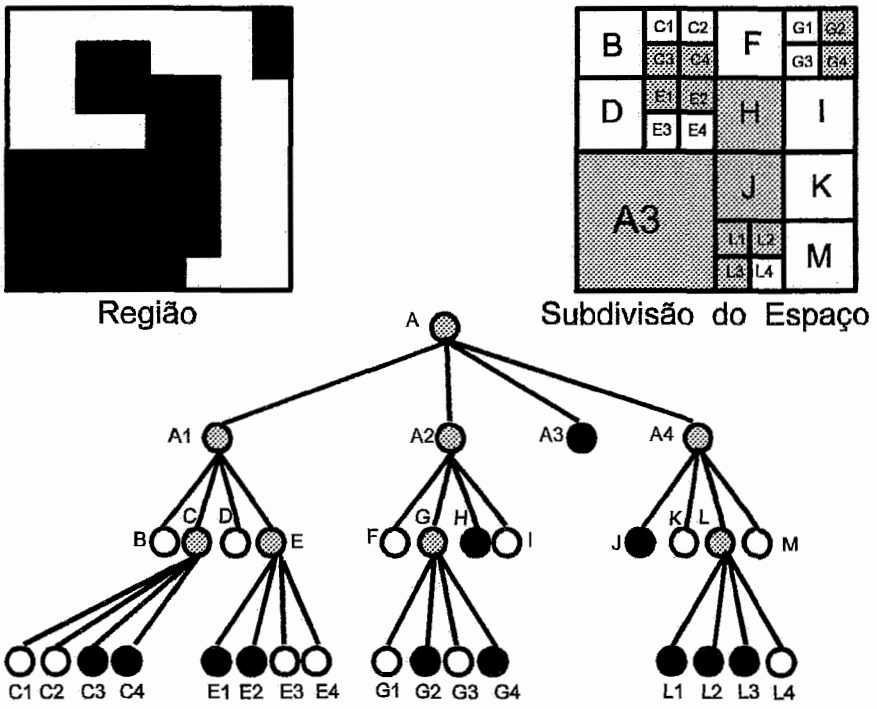


Figura 8 - Representação da *quadtree* como árvore explícita

Se as *quadtrees* forem representadas como árvores de ponteiros, cada nó desta árvore possuirá 5 campos: os quatro primeiros serão ponteiros para os quatro filhos, e o último campo será a cor do bloco (preto, branco ou cinza).

Como foi dito anteriormente, a primeira motivação para o desenvolvimento das *quadtrees* foi a redução do espaço físico no armazenamento de imagens, ao se agregar regiões que possuam uma mesma característica em comum. No entanto, dependendo da imagem quando se representa as *quadtrees* como árvores de ponteiros isto não é sempre verdadeiro. O pior caso acontece quando a imagem é igual a um tabuleiro de xadrez, que em aplicações geográficas provavelmente nunca acontecerá, pois neste caso temos que armazenar a árvore quaternária completa. Para armazenar uma imagem que possua B blocos brancos e P blocos pretos, utilizando *quadtrees* com ponteiros, são gastos $(4(P+B)-1)/3$ nós. Na representação de uma matriz binária de dimensões $2^n \times 2^n$ são necessários 2^{2n}. Conforme aumenta a dimensão da imagem, isto é, o número n cresce, o número de nós da representação binária cresce exponencialmente, enquanto o número de nós da *quadtrees* cresce mais lentamente devido a agregação de blocos com a mesma característica. O número de blocos existente numa *quadtree* é proporcional ao perímetro da imagem. Quando a agregação é miníma a *quadtree* de ponteiros deixa de ser mais eficiente do que a matriz binária[Samet89b].

3.3.3. *Quadtrees* lineares

Como se quer gerenciar grandes massas de informações, fica evidente que não conseguir-se-ia pô-las em memória principal, precisando colocá-las em disco, o que é comum em qualquer SGBD convencional. Portanto, necessita-se de uma representação que possua uma característica seqüencial, e não na forma de árvores com ponteiros (posição aleatória), pois deste modo diminui-se o tempo de acesso a disco (número de *seeks*), além de diminuir o espaço gasto em disco (com a eliminação de ponteiros). Tais estruturas, que mantêm as características espaciais das *quadtrees*, sem a utilização de ponteiros, e ao mesmo tempo são seqüenciais, são conhecidas com *quadtrees* lineares (figura 9).

Por não utilizar ponteiros, as *quadtrees* lineares não são armazenadas como árvores quaternárias. Elas correspondem a uma lista dos nós da *quadtree*, que é obtida calculando para cada bloco um código de localização [Garga82], que o localiza univocamente dentro da *quadtree*. Através da *quadtree* com ponteiros se obtém a *quadtree* linear e vice-versa.

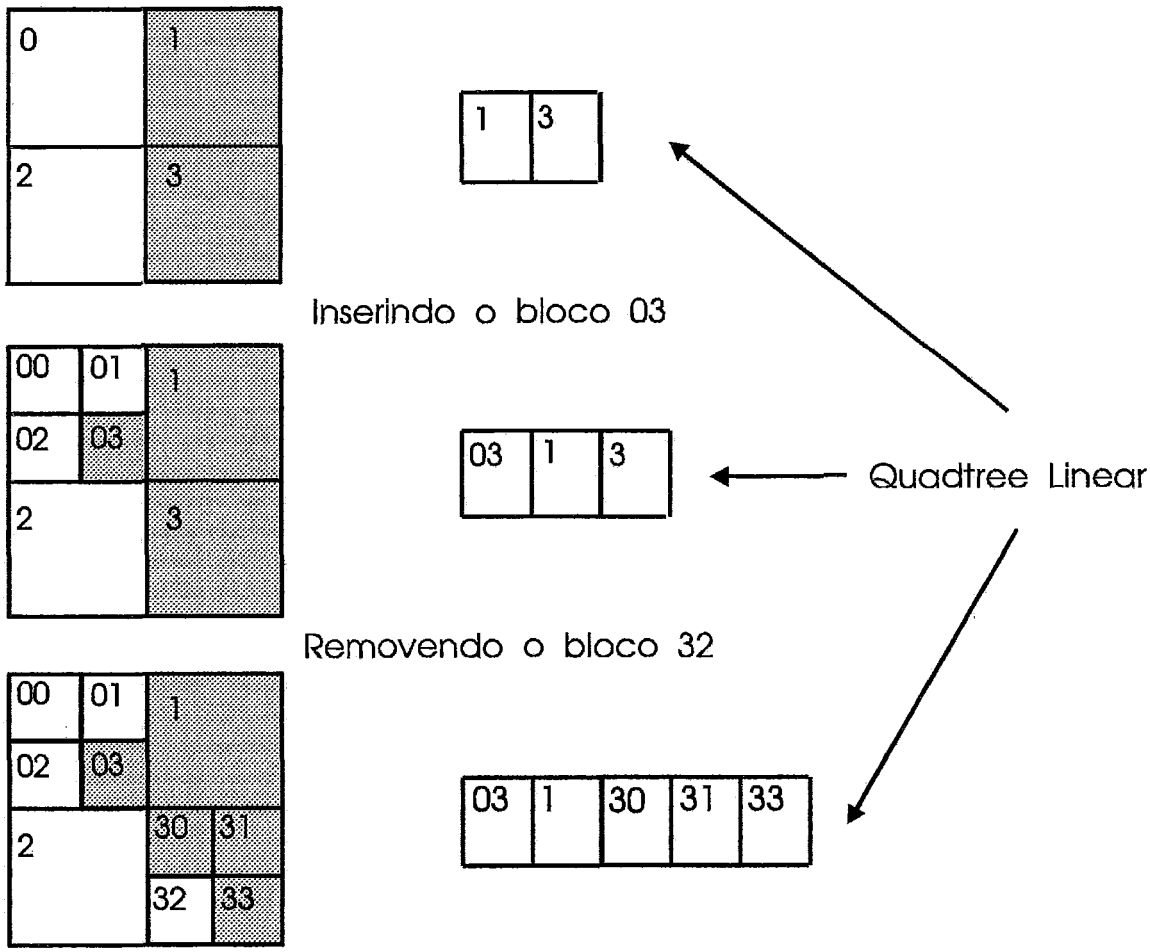


Figura 9 - Representação Linear de uma *quadtree* onde são armazenados apenas os blocos pretos

A lista de blocos pode ser formada de três modos: por todos os blocos da *quadtree* (brancos, pretos e cinzas), pelos blocos brancos e pretos, ou apenas pelos blocos pretos. A primeira opção gasta muito espaço em disco, a última opção é a mais economica em termos de espaço, no entanto, os algoritmos que a manipulam são mais complicados pois os blocos brancos necessitam ser calculados.

3.3.3.1. Código de localização

São utilizados três códigos de localização para a formação da *quadtree* linear: VL (Variable Length), FL (Fixed Length) e FD. Para explicar como esses códigos são calculados será feito o uso da notação explicada a seguir.

Seja uma *quadtree*, com dimensão $2^n \times 2^n$, com raiz no bloco R no nível n, e seja m ($m < n$) o nível de um outro bloco P, cujo código de localização deseja-se

calcular. Define-se a seqüência de blocos $<P_n,P_{n-1},...,P_m>$ de tal modo que $P_n = R$, P_i = PAI(P_{i-1}) para $m < i \leq n$ e $P_m = P$, como a seqüência dos blocos da raiz P_n até P_m. Sejam as direções NO, NE, SO e SE representadas pela função TIPOFILHO4(P_i), que retorna respectivamente 0, 1, 2, 3. O resultado da função TIPOFILHO4 é chamado de código direcional, pois o número obtido representa uma determinada direção. Por exemplo, na figura 8 o bloco *E2* tem valor 1 como resultado para a função TIPOFILHO4.

Usando a seqüência $<y_n,y_{n-1},...,y_m>$, o código de localização y_m é calculado pela relação:

$$\begin{aligned} y_i &= 0 && \text{se} && i = n, \\ y_i &= 4 \,.\, y_{i+1} + \text{TIPOFILHO4}(P_i) && \text{se} && m \leq i < n. \end{aligned} \qquad (1)$$

Por exemplo, o código de localização do bloco *E2* da figura 8 é igual a 13 na base 10 (ou 031 na base 4). Em geral, para uma imagem $2^n \times 2^n$, o código de localização C para um bloco no nível m ($m < n$) pode ser decodificado, através da relação (2), numa seqüência de n-m dígitos $<C_{n-1},C_{n-2},..,C_m>$, onde C_i = TIPOFILHO4(P_i). Para o bloco *E2*, isto resulta em $<C_2,C_1,C_0>$ correspondendo a <NO,SE,NE>.

$$C = \sum_{i=0}^{n-m-1} C_i \cdot 4^i \qquad 0 \leq C_i \leq 3, \qquad 0 \leq C < 4^{n-m}. \qquad (2)$$

A definição dada pela relação (1) corresponde ao código de localização FD, e requer que o nível do bloco seja armazenado junto com o código. O nível do bloco é necessário para se saber onde é o final do processo de decodificação. O zero não pode ser usado porque ele é um dos valores do código direcional. Os códigos de localização VL e FL evitam a necessidade do armazenamento do nível do bloco.

O código VL utiliza a função TIPOFILHO5(P_i) para obter os códigos direcionais, que retorna 1, 2, 3 e 4 respectivamente para NO, NE, SO e SE. Usando a seqüência $<z_n,z_{n-1},...,z_m>$, o código de localização VL, z_m, é calculado pela relação:

$$\begin{aligned} z_i &= 0 && \text{se} && i = n, \\ z_i &= 5 \,.\, z_{i-1} + \text{TIPOFILHO5}(P_i) && \text{se} && m \leq i < n. \end{aligned} \qquad (3)$$

Por exemplo, o código de localização VL do bloco *E2* da figura 8 é igual a 47 na base 10. Usando a relação (4), pode-se decodificá-lo em três dígitos

$<C_2, C_1, C_0>$ que possuem <NO,SE,NE> (ou <1,4,2>) como valores para seu código direcional.

$$C = \sum_{i=0}^{n-m-1} C_i \cdot 5^i \qquad 1 \leq C_i \leq 4, \qquad 0 \leq C < 5^{n-m} \tag{4}$$

Observe que embora o nível não seja mais armazenado explicitamente, este ainda é necessário para que se possa decodificar o código de localização numa sequência com os códigos direcionais apropriados. O nível do bloco é obtido contando-se o número de divisões por 5 que foram realizadas, e subtraindo-se o valor encontrado de n. Para se decodificar o código VL de um bloco B, obtendo o caminho $<C_{n-1}, C_{n-2}, .., C_m>$, onde C_i = TIPOFILHO5(P_i), realizado da raiz até B, divide-se por 5 o código VL do bloco até que o divisor seja igual a 0, os restos encontrados (de baixo para cima) representa o caminho.

O código FL utiliza a função TIPOFILHO4(P_i), a mesma utilizada pelo código FD, sendo calculado do seguinte modo. Cada bloco é representado por uma seqüência de n dígitos $<q_n, q_{n-1}, ..., q_m>$, dada pela relação descrita abaixo. Esta seqüência forma o número $\sum_{i=0}^{n-1} q_i 5^i$, que representa o código de localização do bloco. O dígito q_i representa o código direcional e pode possuir os valores 0,1,2,3,4. O dígito 4 serve para indicar que não há divisão no bloco.

$$\begin{aligned} q_i &= 4 && \text{se} && 0 \leq i < m \\ q_i &= \text{TIPOFILHO4}(P_i) && \text{se} && m \leq i < n \end{aligned} \tag{5}$$

Por exemplo, o bloco $E2$ da figura 8 pode ser codificado pela seqüência $<q_2, q_1, q_0>$ = <0,3,1> ou por 16 na base 10. Do mesmo modo o código de localização FL do bloco H pode ser codificado pela seqüência <1,3,4> ou por 44 na base 10. Para se decodificar o código FL de um bloco B, obtendo o caminho, $<C_{n-2}, C_{n-1}, ..., C_m>$ onde C_i = TIPOFILHO4(P_i), realizado da raiz até B, basta dividir por 5 o código FL do bloco.

Tanto o código FL quanto o código VL utilizam 5 números para sua codificação, o que os torna códigos de base 5. Isto possui o inconveniente de ter que se aplicar operações de divisão ao invés de operações de rotação e módulo, como acontece no código FD.

Quando a lista dos blocos da *quadtree* é formada utilizando-se o código VL, a seqüência obtida leva a uma aproximação sucessiva da imagem, isto é, os blo

cos de nível *m* possuem o código de localização menor do que os blocos com nível *m-1*.

O código FL possui diversas desvantagens em relação ao código VL: o código FL não possui a propriedade apresentada acima, sua decodificação é mais difícil, além de gastar mais espaço em disco. Como vantagem, o código FL faz com que a *quadtree* obtida com sua utilização represente a visita em pós-ordem (também pré-ordem desde que os blocos cinzas não sejam armazenados) quando ordenada em ordem crescente.

O código FD pode ser de tamanho variável ou de tamanho fixo. Se ele for de tamanho fixo, a lista ordenada dos blocos da *quadtree* também representa a visita em pré-ordem aos blocos da *quadtree*.

Para uma imagem $2^n \times 2^n$, cada bloco de tamanho $2^k \times 2^k$ é representado por um código de localização FD de $2n + \lceil \log_2(n+1) \rceil$ bits, onde $\lceil \log_2(n+1) \rceil$ bits são usados para armazenar o nível do bloco. Os $2k$ bits finais dos $2n$ bits do código são normalmente ignorados.

O código FD pode ser obtido através da intercalação dos bits da coordenada do ponto localizado no canto superior esquerdo do bloco, primeiro um bit da coordenada y e depois um bit da coordenada x. Por exemplo, a figura 10 mostra o código de localização componente do código de localização FD (não estão sendo mostrados explicitamente o nível dos blocos) correspondentes dos blocos da figura 8. A ordem dos blocos obtida através desta intercalação é chamada de **ordem de Morton**. A código de Morton foi criado para indexar o espaço bidimensional, e consiste na intercalação dos bits das coordenadas de um ponto pertencente a um plano. Este código não se refere a *quadtrees*, e sim a pontos num espaço bidimensional. Enquanto, o código FD representa a coordenada do ponto superior esquerdo de um bloco de uma *quadtree* acompanhado do nível do bloco. Podemos dizer que o código de Morton faz parte do código FD, pois este é utilizado para representar a coordenada do canto superior esquerdo do bloco. Quando, no código FD, o

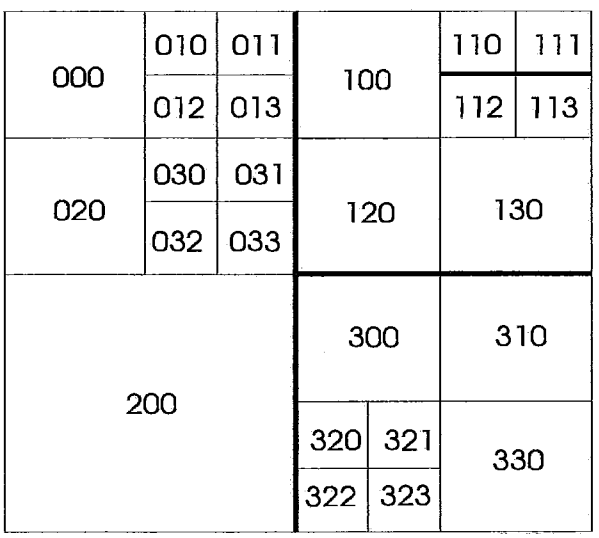


Figura 10 - Exemplo do código FD

nível do bloco é colocado atrás do código de Morton, os blocos ficam ordenados segundo a ordem de Morton.

A *quadtree* é chamada de *quadtree* linear VL quando é formada apenas pelos blocos pretos, e o código de localização usado é o VL.

A representação da *quadtree* como uma coleção de códigos, armazenando apenas os blocos pretos foi apresentada primeiramente por Gargantini [Garga82], que utilizou o código de localização FL e chamou a *quadtree* obtida de *quadtree* linear. Portanto, quando a lista dos blocos é formada apenas pelos blocos pretos, e o código de localização utilizado for o FL, a *quadtree* é chamada de *quadtree* linear FL.

Finalmente, a *quadtree* é chamada de *quadtree* linear FD quando o código de localização utilizado for o código de localização FD.

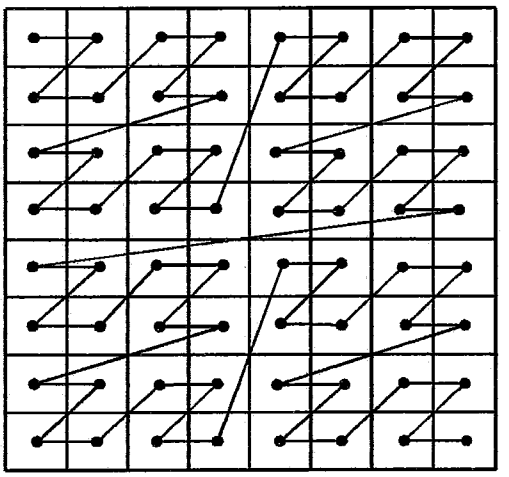


Figura 11 - Código de Morton

3.3.4. Vizinhança de blocos

Um bloco é dito vizinho a um outro quando eles são adjacentes um ao outro. Por exemplo, o bloco *D* da figura 8 é vizinho do bloco *B*, pois seu lado superior (Norte) é o mesmo que o lado inferior (Sul) de *B*.

Um bloco possui, obviamente, 4 lados e 4 vértice. Os quatro lados são chamados de N, S, L, O e os quatro vértices de SE, SO, NE e NO. A relação ADJACENTE(*P*,*Q*,*l*) é definida então do seguinte modo. Dada uma direção *l*, e dois blocos *P* e *Q* que não se sobreponham, a relação ADJACENTE(*P*,*Q*,*l*) será verdadeira se existir um ponto *q* pertencente a *Q* tal que ou *q* é adjacente a uma aresta de *P* na direção *l* ou *q* é adjacente a um vértice *l* de *P*.

A relação de vizinhança não é uma função matemática, pois um bloco pode possuir mais de um vizinho numa dada direção. Por exemplo, o bloco *A3* da figura 8 possui os seguintes vizinhos pelo ladona direção Norte (N): *D*, *E3*, *E4*, *E* e *A1* (os blocos CINZAS também são considerados). Observe que o bloco *A3* compartilha com o bloco *A1* todo o seu lado Norte, enquanto ele compartilha com o seu vizinho *E3* apenas uma parte do seu lado Norte. Isto significa que

para se especificar univocamente um vizinho necessita-se de mais informações sobre ele, como por exemplo, a sua natureza (folha ou não), o seu tamanho e a sua localização. Uma dos modos de especificar uma **função de vizinhança** VIZINHO(*P,l*) que retorna o vizinho *Q* de *P* na direção *l*, é especificando o tamanho de *Q*, do seguinte modo:

VIZINHO(*P,l*) = *Q*: O Bloco *Q* corresponde ao menor bloco (podendo ser CINZA) que é adjacente a aresta *l* do bloco *P*, e cujo tamanho é maior ou igual ao do bloco *P*.

Normalmente, quando se diz que um bloco *Q* é o vizinho de *P* na direção *l*, está se utilizando a função acima, pois caso contrário o critério utilizado deverá ser explicitamente especificado. Esta definição será aplicada nesta tese.

3.3.5. Armazenamento das *quadtrees*

O GDE armazena as *quadtrees* como *quadtrees* lineares FD, onde são armazenados apenas os blocos folhas (brancos e pretos). Foi escolhido o código FD de tamanho fixo por sua decodificação ser mais simples e rápida, além da lista dos códigos dos blocos resultante, quando ordenada, resultar na visita em pré-ordem aos nós da árvore, pois vários algoritmos se utilizam desta ordenação.

Serão utilizados três variações de *quadtrees*, uma para cada tipo espacial. O GDE armazena os pontos, linhas e regiões utilizando respectivamente as *quadtrees: PR quadtree, PMR-f quadtree* e *quadtree* de região. A *PR quadtree* é descrita na seção 3.3.6.1.1, e a *PMR-f quadtree* na seção 3.3.6.2.1

As árvores B [Knut73] [Smith87] são as estruturas ideais para o armazenamento das *quadtrees* lineares FD, pois estas são seqüenciais (como toda *quadtree* linear) e ordenadas. Como muitas vezes é necessário se recuperar toda a *quadtree* foi escolhida a árvore B^+, que por armazenar ponteiros para o nó folha posterior e anterior, possibilita uma busca seqüencial dos códigos dos blocos mais rápida.

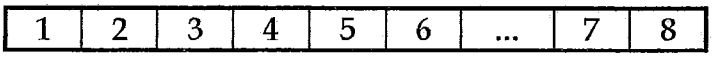
Não foi possível armazenar os blocos da *quadtree* como tuplas de uma relação normal do COPPEREL, pelo fato do COPPEREL utilizar *hashing* para fazer a indexação das tuplas. O que faria com que os blocos das *quadtrees* estivessem distribuídos aleatoriamente no disco, sendo necessários muitos acessos para obtê-los numa ordem seqüencial. Além disto, toda vez que fosse necessário recuperar os blocos de forma ordenada seria preciso ordená-los, pois a maioria dos algorit-

mos utilizados recupera os blocos ordenadamente. O custo desta ordenação seria muito elevado principalmente nos algoritmos de inserção e remoção, pois sempre que fosse inserido ou removido um bloco, a relação (*quadtree*) teria que ser totalmente reordenada, pois o COPPEREL não aproveita a ordem anterior numa reordenação, caso esta tenha sido perdida devido a uma inserção ou remoção.

As árvores B^+ são armazenadas como relações no COPPEREL, sendo cada tupla da relação um nó da árvore B+. Estas árvores B^+ são colocadas pelo GDE, via rotinas de baixo nível do COPPEREL, em relações cujas tuplas representam os blocos da árvore B^+. O tamanho destas tuplas pode ser alterado facilmente pois é parametrizado. Samet utiliza em [Shaff89] blocos para a arvore B+ com tamanho de 1024 bytes.

Estas relações são criadas através de uma chamada à rotina do COPPEREL que cria relações (rotina CRRENT), e não através do comando criar arquivo da LOPEREL. Isto se faz necessário porque estas relações podem possuir um atributo do tipo caracter maior que 127 bytes (nós da arvore B^+), e o analisador semântico do COPPEREL não permite que sejam criadas relações que possuam algum atributo do tipo caracter, cujo tamanho seja maior do que este valor. Além disto, elas são criadas sem que exista nenhum índice sobre o seu único atributo (um caracter de 1024 bytes). Para que a rotina CRRENT identifique estas relações e não aloque para elas espaço para índices, elas são criadas com o tipo 0.

A árvore B^+ possui dois tipos de nós: os nós internos e os nós folhas. Ela armazena as informações associadas às chaves apenas nos nós folhas, deste modo os nós internos funcionam como um índice sobre os nós folhas. Os nós internos e folhas possuem os seguintes campos:



1 - indicador (=1) (2 bytes)
2 - n - número de chaves existentes dentro do nó (2 bytes)
3 - ponteiro para o nó pai (4 bytes)
4 - P_0 - ponteiro para o nó filho, que possui chaves menores que todas as chaves existentes dentro do nó (4 bytes)
5 - C_1 - primeira chave (4 bytes)
6 - P_1 - ponteiro para o nó filho (4 bytes)
7 - C_n - última chave (4 bytes)
8 - P_n - último ponteiro (4 bytes)

Figura 12 - Nó interno da árvore B+

1. indicador (=0) (2 bytes)
2. n - número de chaves existentes dentro do nó (2 bytes)
3. ponteiro para o nó pai (4 bytes)
4. C_1 - ponteiro para o nó folha anterior, que possui chaves menores do que (4 bytes)
5. C_n - ponteiro para o nó folha posterior, que possui chaves maiores do que (4 bytes)
6. C_1 - primeira chave (4 bytes)
7. I_1 - informação associada a chave (tamanho variável)
8. C_n - última chave (4 bytes)
9. I_n - informação associada a chave (tamanho variável)

Figura 13 - Nó folha da árvore B+

A chave da árvore B+ é o código de localização FD, que é armazenado em quatro bytes, os primeiros vinte e oito bits armazenam o componente de localização do código e os últimos quatro bits armazenam o nível do bloco. Deste modo, podemos representar uma imagem $2^{14} \times 2^{14}$ de resolução. Na atual implementação, o nível máximo é nove, o que faz com que o sistema represente imagens de resolução $2^9 \times 2^9$. Esta restrição foi adotada com o objetivo de facilitar a implementação dos comandos de exibição, podendo ser muito facilmente retirada, bastando para tanto alterar o valor de uma constante.

As informações que são associadas a cada bloco dependem do tipo da *quadtree* que está sendo representada. Por exemplo, se estiver sendo representada uma *quadtree* de região, a informação associada é a cor do bloco (branco ou preto) que é representada por 1 byte, conforme veremos na próxima seção. Se for uma *PMR-f quadtree* (para linhas) ou uma *PR quadtree* (para pontos) a informação

ocupa 4 bytes, que representam respectivamente a parte numérica do nome da relação que armazena os segmentos de linha ou o código do ponto contido no bloco (se existir).

Para a manipulação das árvores B^+ foram desenvolvidas rotinas recursivas de inserção, remoção e busca.

Para gerenciar todas as *quadtrees* o GDE se utiliza de uma outra relação chamada QUADTREE, que é descrita na tabela apresentada a seguir:

RELAÇÃO QUADTREE			
Atributo	**Tipo**	**Chave**	**Descrição**
NOME_QUADTREE	inteiro sem sinal	sim	parte inteira do nome da *quadtree*
ORIGEMX	real	não	posição da coordenada X do canto superior esquerdo no sistema cartesiano
ORIGEMY	real	não	posição da coordenada Y do canto superior esquerdo no sistema cartesiano
DESLOCAMENTO	real	não	largura e altura da *quadtree*
TIPO_QUADTREE	inteiro	não	tipo da *quadtree:* 0 = *PR-quadtree* 1 = *PMR-f quadtree* 2 = *quadtree* de região
RAIZ	inteiro	não	número do registro da relação que armazena a *quadtree* que é a raiz da árvore B^+

Tabela 2 - Descrição da relação QUADTREE do GDE.

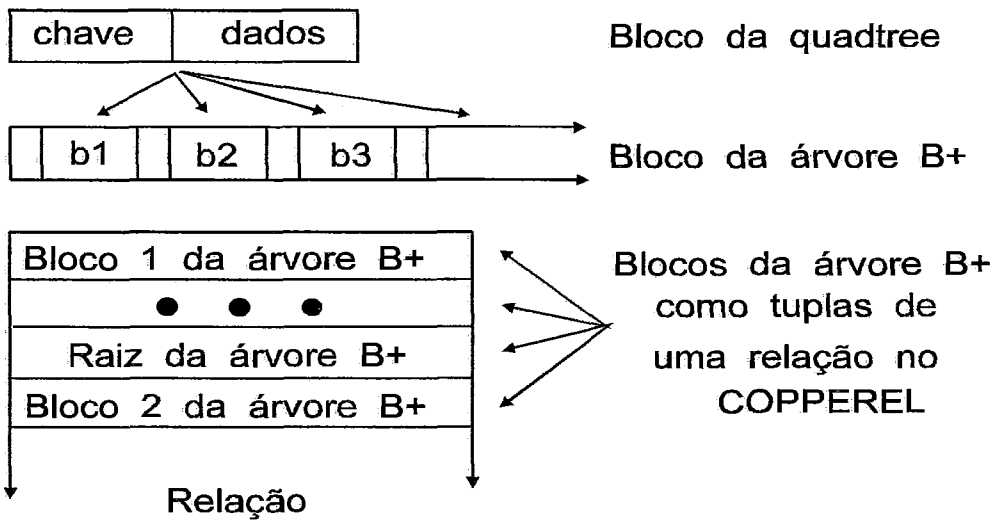


Figura 14 - *Quadtree* como Árvore B+

O GDE acessa a relação QUADTREE através de chamadas ao COPPEREL utilizando a linguagem LOPEREL.

Além da relação QUADTREE o GDE utiliza mais 3 relações: CODIGO_SEGMENTO, CODIGO_PONTO e SURROGATE. Destas, a relação SURROGATE já foi apresentada na seção 3.3.1. A duas primeiras serão apresentadas posteriormente. Todas as quatro relações são criadas pela rotina CRIABASE que é chamada quando o usuário cria uma base de dados. Esta rotina cria estas relações, através de chamadas a rotina CRRENT do COPPEREL, com o tipo -1.

3.3.6. Representação dos dados espaciais

3.3.6.1. Ponto

Os pontos pertencentes a um atributo do tipo ponto de uma relação espacial, são armazenados numa *PR quadtree*. Portanto, se uma relação espacial possuir dois atributos do tipo ponto ela possuirá duas *PR quadtrees*, uma para cada atributo. Desta forma, esta *quadtree* funciona como um índice espacial sobre o atributo do tipo ponto. Para cada ponto pertencente a uma *PR quadtree* é criado um identificador único, isto é, não existirá na *quadtree* dois pontos que possuam o mesmo identificador (código). O número que representa o último código de ponto gerado para cada *PR quadtree* do GDE é armazenado pela relação CODIGO_PONTO. Portanto, esta relação possui uma tupla para cada *PR quadtree* do GDE.

RELAÇÃO CODIGO_PONTO			
Atributo	**Tipo**	**Chave**	**Descrição**
CODIGO	inteiro sem sinal	não	representa o último código gerado da *PR quadtree* representada pelo atributo NOME_QUADTREE.
NOME_QUADTREE	inteiro sem sinal	sim	parte inteira do nome da *PR quadtree*.

Tabela 3 - Descrição da relação CODIGO_PONTO do GDE.

O código de um ponto é associado com sua coordenada através de uma relação armazenada pelo COPPEREL. O nome desta relação é obtido de forma semelhante ao da relação que contém a *quadtree*, sendo que ao invés de ser prefixada a letra 'Q' na frente do número, é prefixada a letra 'P'. Ela possui um atributo

que indica quantos pontos iguais (possuem a mesma coordenada) foram inseridos.

3.3.6.1.1. *PR quadtree*

A *PR-quadtree* é uma adaptação da *quadtree* de região para representar pontos. Ela associa pontos com quadrantes e divide o espaço recursivamente em quatro blocos de mesmo tamanho. A decomposição não é governada pela ordem dados de entrada, isto é, se for modificada a ordem de inserção dos pontos numa *PR quadtree*, a sua decomposição não se altera. Um ponto é armazenado no bloco que o contém. Caso um ponto esteja exatamente entre dois ou mais blocos, é assumido que os lados superior e esquerdo do bloco são fechados, enquanto que os lados inferior e direito do bloco são abertos. Um bloco só poderá conter 0 ou 1 ponto no seu interior (propriedade 1). Além disto, um nó interno da árvore (bloco cinza) só poderá existir se ele possuir no mínimo 2 descendentes que contenham pontos (bloco preto) (propriedade 2). A descrição dos algoritmos de inserção e remoção de pontos numa *PR quadtree* é feito a seguir [Samet89a].

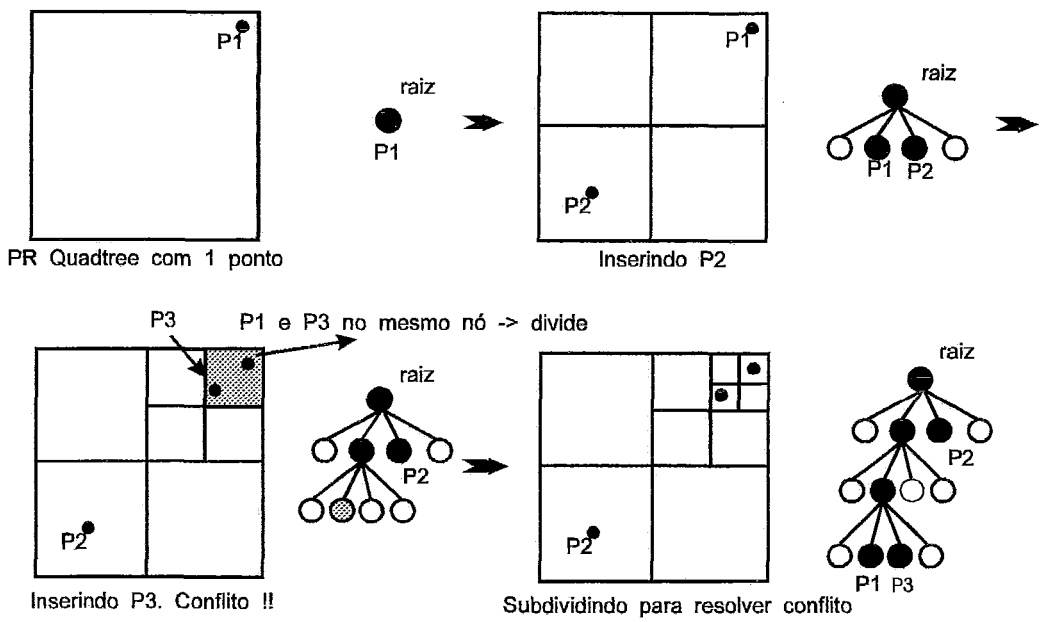


Figura 15 - Inserção de pontos numa *PR quadtree*

Para exemplificar o algoritmo de inserção de um ponto numa *PR quadtree* utilizaremos a figura 15. Nela, os blocos serão referenciados pelo caminho percorrido da raiz até eles. Ela mostra a inserção de 3 pontos (*P1*, *P2* e *P3*) numa *PR quadtree* inicialmente vazia. Primeiramente insere-se o ponto *P1*. Como não

existe nenhum outro ponto dentro do bloco que contém *P1* (bloco *Raiz*) não é necessário dividí-lo. Ao se tentar inserir o ponto *P2* encontra-se um conflito, pois o bloco *Raiz* já possui o ponto *P1*. Logo, dividi-se o bloco e insere-se os pontos *P1* e *P2* nos blocos filhos que os contém. No caso, o filho NE (bloco <NE>) conterá o ponto *P1*, e o filho SO (bloco <SO>) conterá o ponto *P2*. Finalmente, ao se tentar inserir o ponto *P3*, encontra-se novamente conflito pois o bloco <NE>, que contém o ponto *P3*, já possui o ponto *P1* em seu interior. Divide-se então o bloco <NE>. Os pontos *P1* e *P3* estão ambos localizados no bloco <NE,NE>. Portanto, é necessário que se divida também o bloco <NE,NE>. Neste momento a inserção do ponto *P3* termina pois os pontos *P1* e *P3* estão localizados em blocos distintos.

Este algoritmo é recursivo pois ao inserir um ponto num bloco já ocupado temos que dividí-lo, e continuar dividindo até que o ponto seja inserido num bloco não ocupado. Pois, pela definição da *PR-quadtree* apresentada acima, um bloco **não** pode conter mais de 1 ponto no seu interior.

Para remover um ponto de uma *PR-quadtree*, primeiramente procura-se pelo bloco que o contém, caso o ponto exista, este é removido do bloco. Após a remoção, verifica-se quantos irmãos pretos e cinzas ele possui. Se ele possuia mais do que 1 irmão preto e,ou, cinza (bloco subdividido), a remoção termina. Caso contrário os 4 irmãos devem ser unidos, evitando assim que a *PR quadtree* viole a propriedade 2, e aplica-se recursivamente o mesmo processo ao bloco resultante da união. Uma outra maneira de explicar o algoritmo é vendo a *PR quadtree* como uma árvore. Neste caso após remover um ponto da *PR quadtree* de um nó da árvore, tem-se que subir para o pai do nó e ver quantos descendentes pretos ele possui, se ele possuir mais de 1 descendente preto não será necessário unir os seus filhos. Caso contrário, seus filhos serão unidos e ele possuirá, caso exista, o único ponto encontrado.

O algoritmo de remoção é exemplificado na figura 16. Os blocos desta figura serão referenciados através do caminho percorrido da raiz até eles, por exemplo, o bloco onde se encontra o ponto *P1* será referencidado por <NE,NE,NE>.

Na figura 16, primeiramente remove-se o ponto *P2*, que se encontra no bloco <SO>. Olhando os irmãos deste bloco vê-se que o seu irmão <NE> é cinza. Logo não é necessário unir os 4 irmãos. Na remoção do ponto *P3* do bloco <NE,NE,SO>, é necessário unir os 4 irmãos, pois o bloco <NE,NE,SO> (onde foi removido *P3*) só possui 1 irmão preto e nenhum irmão cinza. Após união, o bloco <NE,NE>, que era subdividido (cinza), passa a ser preto, isto é, passa a conter o ponto *P1*. Agora aplica-se o mesmo processo ao bloco <NE,NE>,

gerando o bloco preto <NE>, e aplicando-se novamente ao bloco <NE> obtém se o bloco raiz.

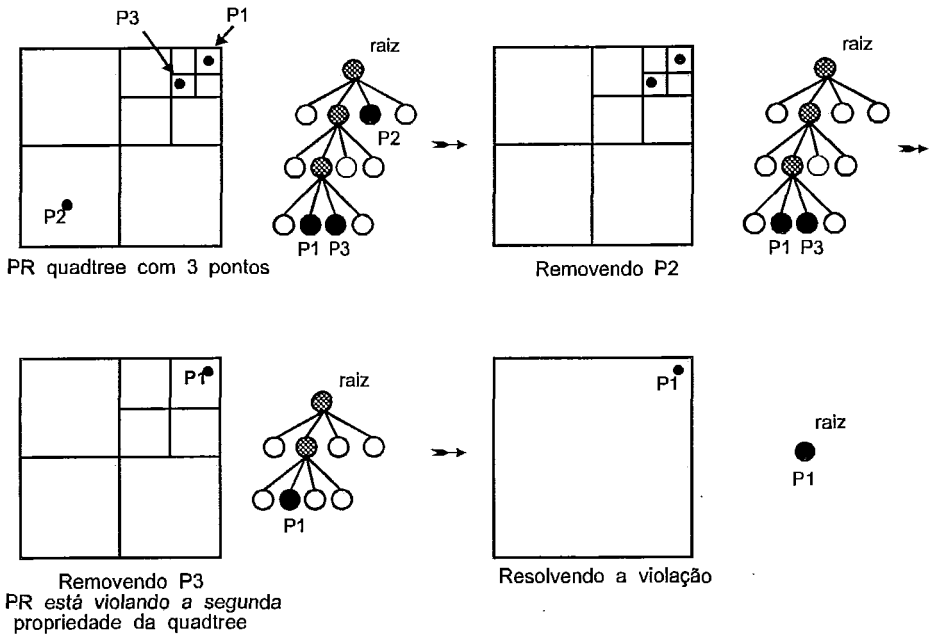


Figura 16 - Remoção de pontos na *PR quadtree*

3.3.6.2. Linha

Cada valor de um atributo do tipo linha é armazenado numa *PMR-f quadtree* própria. Portanto, uma relação espacial que possua m atributos do tipo linha e n tuplas, possuirá $m \times n$ *PMR-f quadtrees*, sendo m *PMR-f quadtrees* para cada tupla.

A cada segmento de linha é associado um código, através de uma relação. O número que representa o último código gerado para cada *PMR-f quadtree* do GDE é armazenado pela relação CODIGO_SEGMENTO. Portanto, esta relação possui uma tupla para cada *PMR-f quadtree* do GDE.

RELAÇÃO CODIGO_SEGMENTO			
Atributo	**Tipo**	**Chave**	**Descrição**
CODIGO	inteiro sem sinal	não	representa o último código gerado da *PMR-f quadtree* representada pelo atributo NOME_QUADTREE.
NOME_QUADTREE	inteiro sem sinal	sim	parte inteira do nome da *PMR-f quadtree.*

Tabela 4 - Descrição da relação CODIGO_SEGMENTO do GDE.

O código de um segmento é associado com suas coordenadas através de uma relação armazenada pelo COPPEREL. O nome desta relação é semelhante ao da relação que contém a *quadtree*, só que ao invés da letra 'Q' é prefixada a letra 'S'. Assim, são armazenados na *quadtree* os códigos dos segmentos, e não a sua coordenada, simplificando os algoritmos de inserção e remoção de segmentos de linha.

3.3.6.2.1. *PMR-f quadtree*

Existem diversas estruturas de dados que podem ser utilizadas para representar segmentos de linhas. Dentro da família das *quadtrees* existem vários tipos diferentes que podem ser usados [Samet85b]. Por exemplo, tem-se: *edge quadtree, segment quadtree, PM1 quadtree, PM2 quadtree, PM3 quadtree, PMR quadtree* e *PMR-f quadtree*. Algumas destas *quadtrees* podem ser facilmente implementadas como *quadtrees* lineares (ex. *PMR* e *PMR-f quadtree*) enquanto outras não (ex. *PM1* e *PM2 quadtree*).

A *PMR-f (PM Random fragment) quadtree* [Samet89a] representa as linhas, perfeitamente, isto é, não é feita nenhuma aproximação. As linhas são representadas por segmentos de linha, sendo que um segmento é definido através de dois pares de coordenadas. As atualizações na *PMR-f quadtree* são consistentes, quando um segmento de linha é removido, o banco de dados pode retornar ao estado anterior (sem aproximação) de quando o segmento não tinha sido removido.

A *PMR-f quadtree* é uma adaptação da *PMR quadtree* [Shaff89] para armazenar fragmentos de linhas, que são pedaços de segmentos, e ocorrem quando são removidos partes de segmentos de linha, como será visto a seguir. Primeiramente será descrita a *PMR quadtree*, e posteriormente, através da introdução do conceito de fragmento de linha será descrita a *PMR-f quadtree*.

A *PMR quadtree* é considerada um método de *bucket*, sendo que o termo *bucket* possui um significado diferente do usado normalmente em computação. Ela é dita como sendo um método de *bucket* porque ela divide um bloco quando o número de segmentos em seu interior for maior do que um número n, onde n é chamado de capacidade do *bucket*. Observe que isto não impede que um bloco possua mais do que n segmentos. Com isto, a *PMR quadtree* consegue armazenar sem dificuldades vértices de qualquer grau. No GDE a capacidade do *bucket* (n) é igual a 4, esta escolha foi feita baseada nos resultados apresentados por Samet [Samet85]. A figura 17 mostra a construção de uma *PMR quadtree* com capacidade de *bucket* igual a 2.

Um segmento é inserido em todos os blocos que intercepta. Na inserção do segmento em um bloco, testa-se o bloco, após a inclusão, se este possuirá mais segmentos do que a capacidade de *bucket*. Caso afirmativo, o bloco é dividido uma única vez em quatro quadrantes iguais.

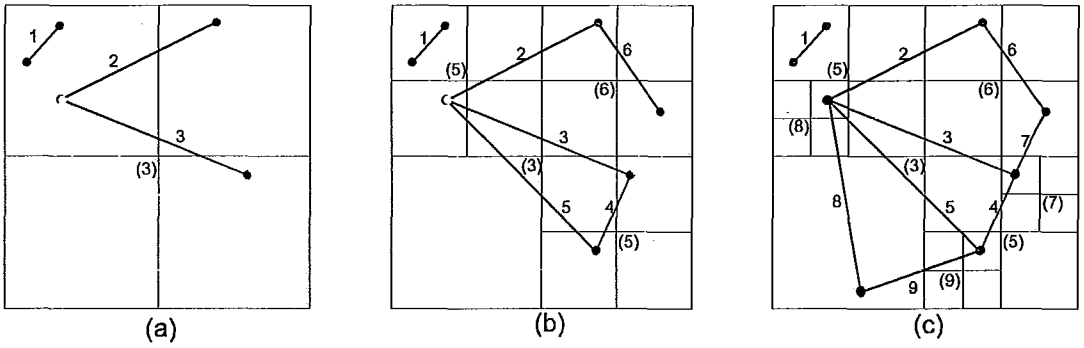


Figura 17 - O processo de construção de uma *PMR quadtree* com capacidade de *bucket* igual a 2; (a) Mostra *quadtree* após a inserção dos segmentos 1-3. A *quadtree* foi dividida uma única vez quando o segmento 3 foi inserido, conforme está indicado pelo número entre parenteses. (b) Mostra a *quadtree* após a inserção dos segmentos 4-6. Três blocos foram divididos. (c) Mostra a *quadtree* após a inserção dos segmentos 7-9. Mais três blocos foram divididos.

Por outro lado, quando um segmento de linha é removido, o será de todos os blocos que este intercepta. Após a remoção de um segmento de um bloco, é testado se o bloco junto com os seus três irmãos possuem menos segmentos que um determinado valor s (limite de divisão). Caso afirmativo, o bloco mais os seus três irmãos são unidos, e o processo de união será reaplicado ao bloco resultante da união e aos seus irmãos. No GDE o limite de divisão é igual a dois, isto é, quatro blocos irmãos serão unidos se eles juntos possuírem 0 ou 1 segmento de linha.

Observe que, a capacidade do *bucket* (n) e o limite de divisão (s) são dois conceitos distintos. O primeiro é utilizado no processo de inserção de um segmento de linha em um bloco, enquanto o segundo é utilizado no processo de remoção de um segmento de linha de um bloco.

Portanto, para organizar os dados dinamicamente, a *PMR quadtree* usa um par de regras, uma para a divisão e outra para a união dos blocos. A primeira é chamada toda vez que é acrescentado um segmento de linha a um bloco, e diz que um bloco tem que ser dividido, uma única vez, quando ele possuir mais de n segmentos. Note que esta regra não garante que o bloco não possua mais de n segmentos no seu interior. Por sua vez, a regra de união é chamada toda vez

que um segmento é removido de um bloco. O bloco é unido com os seus irmãos, se eles (os quatro irmãos) possuírem menos que s segmentos. Este processo é reaplicado recursivamente ao nó resultante da união e aos seus irmãos.

Como foi dito anteriormente, a *PMR-f quadtree* é uma extensão da *PMR quadtree* para manipular fragmentos de linhas, que são pedaços de segmentos. Nas aplicações geográficas, os fragmentos aparecem quando é feita a interseção de um mapa de linhas com uma área. Como as bordas da área provavelmente não irão coincidir com os pontos extremos do segmento, certos segmentos serão cortados. A figura 18 apresenta esta situação. Os pontos gerados pela interseção da área com o segmento são chamados de pontos de corte, e os pedaços de segmentos resultantes são os fragmentos. Se alterássemos os pontos extremos do segmentos para refletir o corte, estaríamos então gerando um novo segmento, tornando assim improvável o retorno da base de dados ao estado anterior, caso fosse inserido o pedaço (fragmento) removido. Isto porque, no espaço contínuo podemos sempre representar o novo segmento como sendo colinear com o segmento original. No entanto, no espaço discreto isto não será sempre possível, porque as coordenadas contínuas do ponto de interseção podem não corresponder a nenhuma coordenada do espaço discreto.

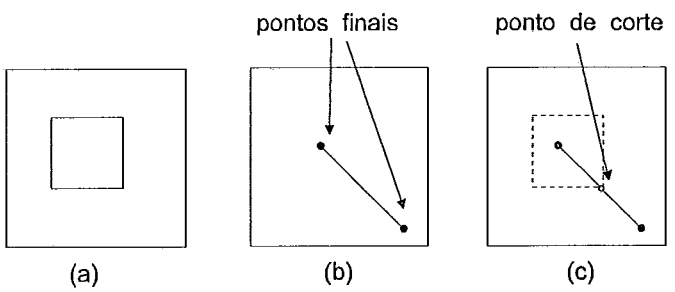


Figura 18 - Pontos de Corte - Fragmento de linha (C) resultante da interseção do bloco (A) com o segmento de linha (B)

Uma alternativa é não alterar a descrição dos segmentos, e utilizar a propriedade espacial das *quadtrees* para especificar quais os pedaços do segmento de linha estão representados. Deste modo, um bloco referenciará um segmento mesmo se o segmento inteiro não existir, isto é, se só existir alguns fragmentos dele. Por exemplo, na figura 19 (c) representa-se a interseção da região mostrada em (a) com os segmentos mostrados em (b). Observe, que em (c) os segmentos 1 e 2 não existem mais em sua totalidade, o que existe é um pedaço do segmento 1 e um pedaço do segmento 2. O bloco NO contém uma referência ao segmento 1, e não a um novo segmento que representa o pedaço do

segmento 1 que ainda existe. Deste modo, para se saber qual parte do segmento que esta sendo representado deve-se recortá-lo com a borda do bloco onde o segmento é encontrado, pois este pode não estar sendo totalmente representado. Esta é a solução utilizada na *fragment quadtree* [Samet89a]. A *PMR-f quadtree* é a *PMR quadtree* com a incorporação da noção de fragmento no seu princípio de decomposição.

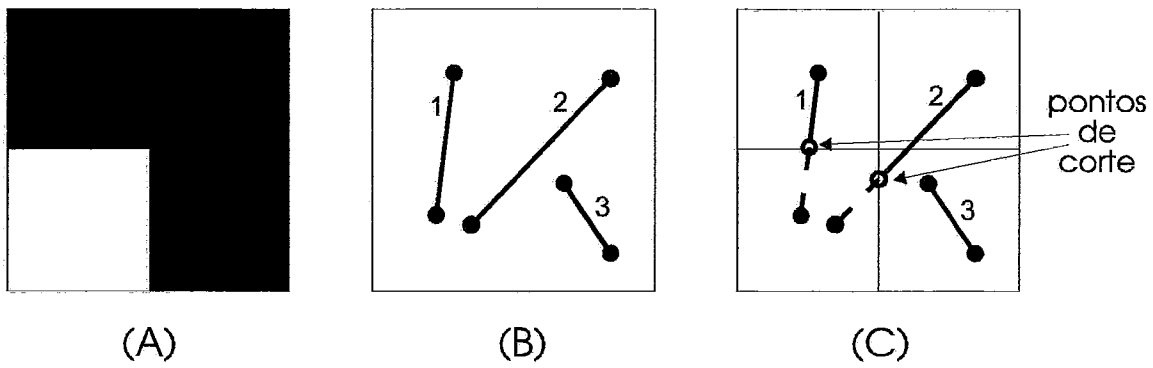


Figura 19 - Fragmentos de Linha

A inserção de um fragmento *F*, representando a interseção de um segmento *S* com um bloco *B*, é então realizada do seguinte modo. Se *B* corresponder a um nó folha (figura 20(a) e 20(b)), então *F* é inserido em *B* e é verificado então se o bloco *B* necessita ser unido. Isto acontecerá se a inserção de *F* fizer com que os quatro irmãos possuam os mesmos fragmentos, e o número de segmentos existentes for menor do que *s* (limite de divisão) (figura 20(b)). Se *B* não existir ainda na *quadtree*, isto é ele faz parte de um bloco maior (branco ou preto) *B1* (figura 20(c)), então o bloco *B1* é dividido até que o bloco *B* exista. Insere-se então o fragmento *F*, não sendo necessário testar se o bloco *B* precisa ser unido. Se *B* representar um bloco cinza então os fragmentos correspondentes a interseção de *L* com os filhos de *B* são inseridos recursivamente (figura 20(d)).

A remoção de um fragmento *F* que representa a interseção de um segmento *S* com um bloco *B* é realizada do seguinte modo. Se *B* for um bloco folha que contém *F* (figura 21(a) e 21(b)), então *F* é removido de *B*, e é testado se *B* precisa ser unido com os seus irmãos. Se *B* fizer parte de um bloco maior *B1* (figura 21(c)), então *B1* é dividido até que *B* exista, e é removido o segmento *S* de *B*. Se *B* corresponder a um bloco cinza (figura 21(d)), então são removidos todas as referências ao segmento *S* dos filhos de *B*.

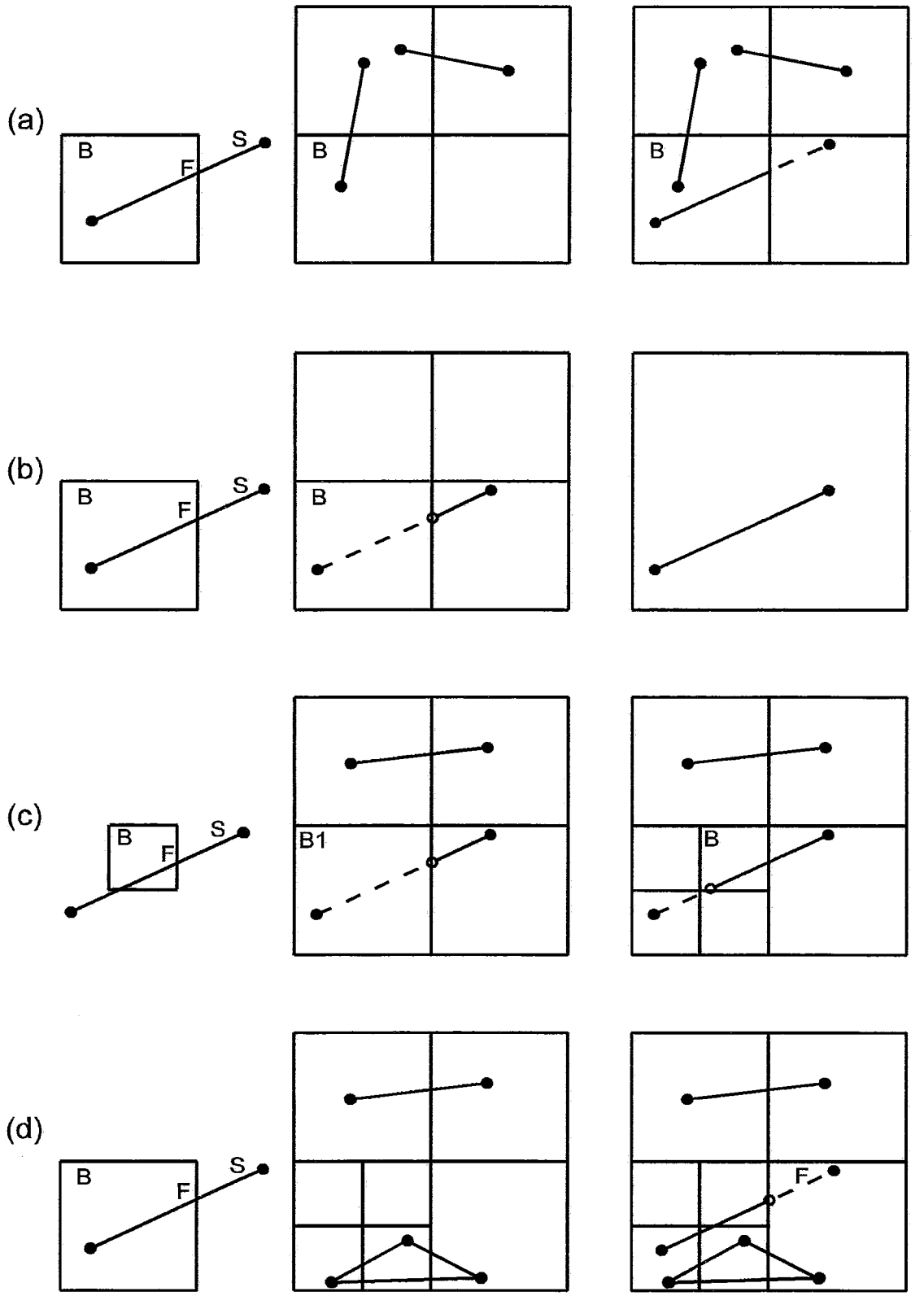


Figura 20 - Inserção de fragmentos de linha (F) que representam a interseção de um segmento (S) com um bloco (B) numa *PMR-f quadtree* com capacidade de *bucket* igual a 2; (a) e (b) o bloco B corresponde a um nó folha; (b) o segmento (S) é recomposto. Os quatro irmãos são unidos. (c) O bloco B não existia na *quadtree;* (d) o bloco B corresponde a um bloco cinza.

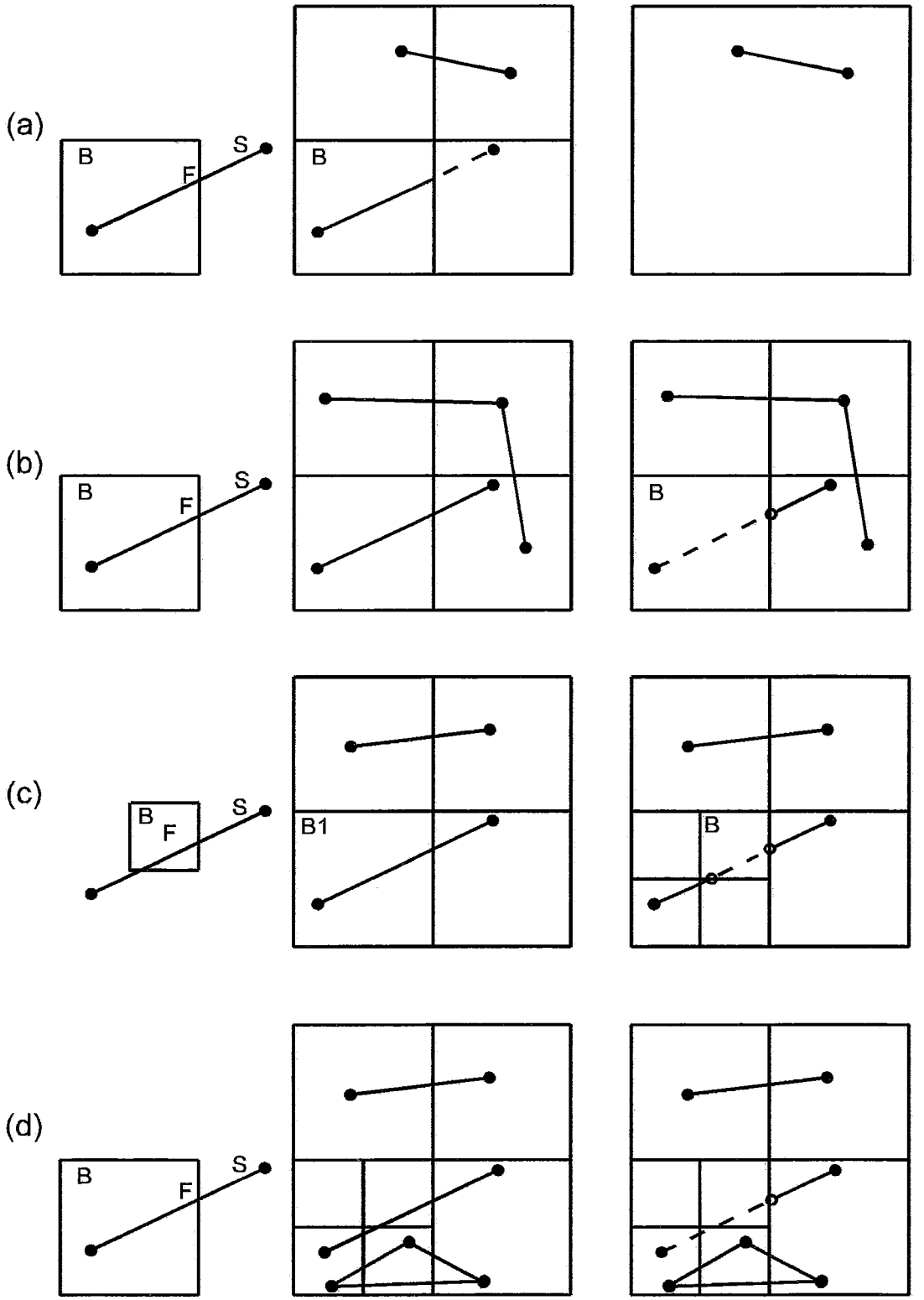


Figura 21 - Remoção de fragmentos de linha (F) que representam a interseção de um segmento (S) com um bloco (B) numa *PMR-f quadtree* com limite de divisão igual a 1; (a) e (b) o bloco B corresponde a um nó folha; (a) o segmento (S) é totalmente removido, e o número de segmento dos quatro irmãos é igual ao limite de divisão. Os quatro irmãos são unidos. (c) O bloco B não existia na *quadtree;* (d) o bloco B corresponde a um bloco cinza.

Um bloco necessitará ser unido quando o número de segmentos distintos existentes nele e em seus irmãos for menor do que s, e os fragmentos forem contínuos no bloco produzido com a união. Fragmentos cujos blocos podem ser unidos desta maneira são ditos compatíveis, enquanto os que não podem são ditos incompatíveis, pois pode ocorrer dos quatro irmãos possuírem menos que s (s=2) segmentos distintos, mas os fragmentos não forem compatíveis, isto é, existem um ponto de corte sendo representado. Na figura 22 os blocos contidos em (A) são compatíveis enquanto os blocos contidos em (B) são incompatíveis.

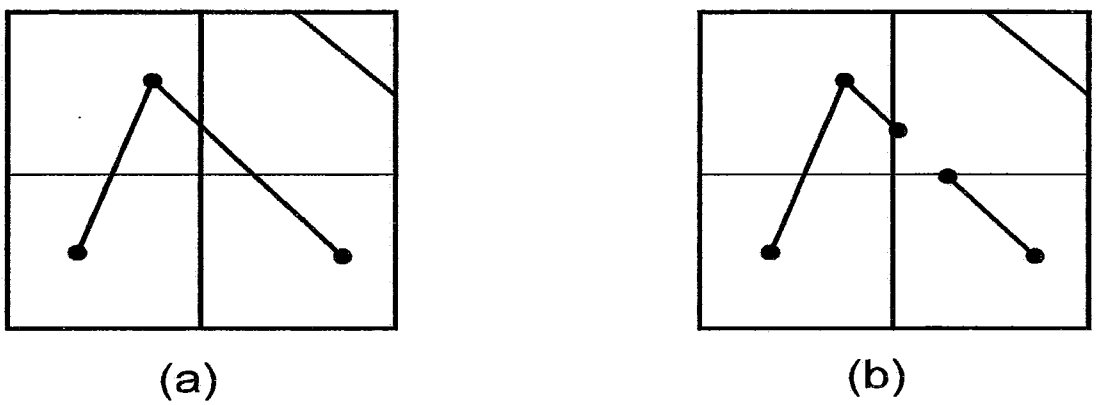


Figura 22 - Compatibilidade de segmentos

Um bloco, agora, é dividido até que este não contenha nenhum ponto de corte no seu interior, e será dividido mais uma vez caso possua mais do que *n* segmentos.

Como a *PMR-f quadtree* possui mais de 1 segmento por bloco, é necessário ter uma maneira de se implementar registros de tamanho variável. Como elas são armazenadas usando *quadtrees* lineares, o modo mais natural é se duplicar o código de localização do bloco, fazendo com que a árvore B^+ que armazena a *quadtree* possua chaves repetidas.

3.3.6.3. Região

As regiões são representadas utilizando-se a *quadtree* de região, apresentada no início do capítulo. A *quadtree* linear possui para cada bloco, um campo de 32 bits para armazenar o código de localização do bloco e um byte para armazenar a cor do bloco.

3.3.7. Funções Implementadas

O GDE realiza a interseção e a união das *quadtrees*. A união de *quadtrees* é realizada somente se elas forem do mesmo tipo, isto é, ele une uma *PMR-f quadtree* com outra *PMR-f quadtree*, mas não une uma *PMR-f quadtree* com uma *PR quadtree*. A interseção é realizada sobre *quadtrees* do mesmo tipo, e sobre uma *quadtree PR* ou *PMR-f* com uma *quadtree* de região, resultando respectivamente numa *PR* ou *PMR-f quadtree*. A operação de interseção é muito utilizada nos sistemas geográficos pois através dela pode-se recuperar informações que pertençam a uma dada região (recuperação por janela).

Além da interseção e da união o GDE compara duas *quadtrees* do mesmo tipo e retorna se elas são iguais ou não. Esta função foi implementada para que o GEOCOPPE pudesse suportar os operadores relacionais sobre os atributos espaciais. Por exemplo, quando se faz a união de duas relações é necessário saber também se os atributos espaciais são iguais ou não, para se poder eliminar as tuplas repetidas.

Foram implementadas três funções para análise dos dados espaciais: área e perímetro de regiões (*quadtree* de região) e comprimento de linhas (*PMR-f quadtree*).

Para a entrada das regiões no GEOCOPPE foi implementado um algoritmo para conversão de imagem *raster* para *quadtree*. Deste modo, as regiões são fornecidas através de arquivos no formato *raster*. O algoritmo utilizado [Shaff87] é muito eficiente, pois só insere os blocos na *quadtree* quando estes são maximais, evitando assim, que seja necessário realizar futuras uniões de blocos. Outros algoritmos de conversão de *quadtrees* podem ser encontrados em [Samet80] [Dyer80] [Samet80b].

3.3.7.1. Operações de Conjunto (interseção/união)

As operações de conjunto sobre *quadtrees* que possuem a mesma origem, chamadas de *quadtrees* alinhadas, e o mesmo tamanho de mapa são muito fáceis de serem implementadas [Samet85]. Alguns algoritmos de operações de conjunto sobre *quadtrees* lineares alinhadas podem ser encontrados em [Bauer85]. O GDE realiza a interseção/união de duas *quadtrees*, mesmo que estas possuam origens diferentes e tamanho de mapas diferentes, porém com o mesmo tamanho de grid.

O algoritmo utilizado pelo GDE é baseado no algoritmo apresentado em [Samet85] para interseção de duas *quadtrees* desalinhadas, mas com o mesmo tamanho para o grid e para o mapa.

Será apresentado o algoritmo de interseção de uma *quadtree* de região *QUAD1* com uma outra *quadtree* de região *QUAD2* resultando numa nova *quadtree* *QUAD3*. O algoritmo de união e os algoritmos entre *quadtrees* de tipos diferentes são obtidos facilmente a partir do algoritmo de interseção. No início da rotina CONJUNTO, *QUAD3* é criada de modo a ser alinhada com *QUAD1*, possuindo portanto, a mesma origem e o mesmo deslocamento que *QUAD1*.

Para cada bloco, *B1*, de *QUAD1*, procura-se em *QUAD2* pelos blocos, *B2*, que sobreponham *B1*. Calcula-se então o maior bloco possível, *B3*, gerado pela interseção de *B1* com o bloco, *B2*, de *QUAD2*. A seguir aplica-se a operação de conjunto sobre *B1* e *B2*. No caso, como a operação de conjunto é a interseção, o bloco *B3* será preto se *B1* e *B2* o forem, e será branco caso um dos dois blocos seja branco. Para que os blocos de *QUAD3* sejam escritos no disco apenas uma vez é utilizado o vetor *tabsai*, que armazena os blocos de *QUAD3* antes de escrevê-los em disco. Um bloco só é escrito em disco quando se tem certeza de que ele não pode ser unido com outros irmãos seus. Por exemplo, a figura 23(a) mostra a interseção de um bloco *B1* de *QUAD1* com *QUAD2*. A figura 23(b) apresenta a decomposição realizada no bloco *B1*, mostrando todos os blocos maximais obtidos. Note que se os blocos 10, 11, 12 e 13 da figura 23(b) são todos brancos portanto eles necessitarão ser unidos. No entanto, isto só será descoberto após o processamento do bloco 13. Utiliza-se o vetor *tabsai* para evitar que os blocos 10, 11 e 12 sejam escritos em *QUAD3* desnecessariamente.

Como os blocos de *QUAD3* são gerados pela ordem de Morton (pois os blocos de *QUAD1* são sempre lidos pela ordem de Morton), o segundo, terceiro e quarto sub-quadrantes de um bloco no nível n só serão processados depois que os sub-quadrantes anteriores o forem, isto é, o quadrante SO não será processado antes que os quadrantes NO e NE o forem. Os blocos vão ser retirados de *tabsai* e inseridos em *QUAD3* quando se estiver processando um bloco SE, e verificar-se que este e os seus três irmãos (que se encontram em *tabsai*) possuem cores diferentes. Se os blocos possuírem a mesma cor estes serão unidos e retirados de *tabsai*, o bloco resultante da união é colocado no topo de *tabsai*. Logo, necessariamente quando se processa um bloco SE retira-se dois ou três blocos de *tabsai*. Portanto, o número máximo de blocos que podem existir em *tabsai*, numa *quadtree* de dimensões $2^n \times 2^n$, é $3n + 1$.

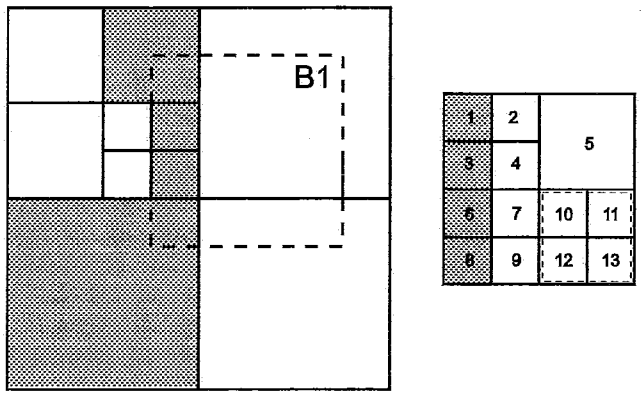


Figura 23 - Interseção do bloco *B1* com uma imagem; (a) Interseção do bloco *B1* (em linhas tracejadas) com uma imagem; (b) Decomposição e a ordem de processamento do bloco *B1* como dirigida pela decomposição da imagem. As linhas tracejadas indicam que os blocos constituintes irão ser unidos para formar um bloco maior.

Para que os blocos de *QUAD2* sejam lidos apenas uma única vez utiliza-se dois vetores, *xaresta* e *yaresta*, para armazenar os blocos ativos de *QUAD2* (isto é, os blocos que foram lidos, mas não foram totalmente processados). O vetores *xaresta* e *yaresta* armazenam os nós ativos de *QUAD2*. Primeiramente, considera-se, em qualquer instante, a borda dos nós de *QUAD1* que foram processados. Como os nós de *QUAD1* são lidos na ordem de Morton, a borda ativa terá a forma de uma escada (ver figura 24). A borda ativa, quando ela cruza um mapa de dimensão $2^n \times 2^n$, forma segmentos horizontais e verticais cuja soma do comprimento, de todos os segmentos verticais e de todos os segmentos horizontais separadamente, é 2^n pixels. Os nós ativos de *QUAD2* serão aqueles que, num dado instante, cruzam a borda ativa de *QUAD1*..

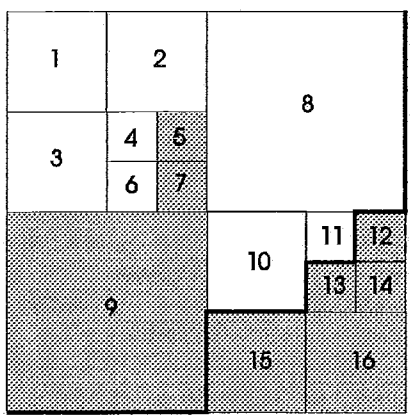


Figura 24 - Borda Ativa após o processamento do bloco 11. A linha tracejada mostra a mudança da borda ativa após o processamento do bloco 12.

Para cada bloco *B1* de *QUAD1*, chama-se a rotina CONJ. Esta rotina processa o bloco *B1* em pedaços, começando pelo seu canto superior esquerdo, cuja coordenada é *(cx1,cy1)*, até o canto inferior direito do bloco. A divisão de *B1* é determinada pelo tamanho e pela posição dos blocos de *QUAD2* que correspondem aos pixels cobertos por *B1* (ver figura 23). Para cada um desses pedaços do bloco *B1*, os vetores *xaresta[cx1]* e *yaresta[cy1]* são consultados para determinar se o bloco de *quad2* que eles armazenam contém o ponto *(cx1,cy1)*. Se apenas um dos dois vetores contiver o bloco que contém *(cx1,cy1)*, o bloco é copiado para o vetor deficiente. Se nenhum dos dois vetores contiver a descrição de um bloco que contenha o pixel *(cx1,cy1)*, este bloco é então lido de *QUAD2*, e é colocada sua descrição em *xaresta[cx1]*e em *yaresta[cy1]*. Este é o único tipo de atualização necessária nos dois vetores, para garantir que os blocos de *QUAD2* sejam lidos no disco apenas uma única vez.

Para entender que isto é realmente verdade, observe o modo com que os blocos (ou pedaços do bloco) de *QUAD1* são processados. Num dado momento, apenas os pixels do vértice formado por dois segmentos da borda ativa podem ser processados. Assumindo que após processar um bloco localizado em *(cx1,cy1)* e com largura *largura*, tanto *xaresta[cx1]* quando *yaresta[cy1]* possuem a descrição do bloco correto, a nova borda ativa irá formar no máximo duas novas posições de ângulos, uma em *(cx1,cy1+largura)* e outra em *(cx1+largura, cy1)*. Quando o pixel *(cx1,cy1+largura)* é processado, *xaresta[cx1]* ainda conterá a descrição do mesmo bloco, pois nenhum outro pixel com o valor da coordenada X igual a *cx1* foi processado neste meio tempo. O mesmo raciocínio se aplica ao pixel *(cx1+largura,cy1)* e a *xaresta[cy1]*.

Busca dos blocos da *quadtree* *QUAD2*

Para se recuperar o bloco de *QUAD2* que contenha um pixel de *QUAD3*, *(cx1,cy1)*, a rotina CONJ chama a rotina ACHA_BLOCO. Como as *quadtrees* possuem tamanho de mapas e origens diferentes é necessário que se converta *(cx1,cy1)* para o sistema de coordenadas de *QUAD2*. Isto é feito em duas etapas, na primeira o pixel *(cx1,cy1)* é convertido para o sistema de coordenada global na coordenada *(XG,YG)* através da seguinte equação:

$$XG=\frac{CX1}{2^n}d+XO$$

$$YG=\frac{CY1}{2^n}d+YO$$

onde

d = largura/altura da *quadtree* QUAD1
n = nível máximo da *quadtree* (no GDE = 14)
XO = coordenada X da origem da *quadtree* QUAD1
YO = coordenada Y da origem da *quadtree* QUAD1

Se a coordenada *(XG,YG)* não estiver contida no quadrado representado por QUAD2, a rotina devolve um bloco em branco. Caso contrário, é executada a segunda etapa da conversão. Nela a coordenada *(XG,YG)* é convertida para a coordenada *(CX2,CY2)* do sistema de coordenadas da *quadtree* QUAD2, através da seguinte equação:

$$CX2=\left|\frac{XG-XO}{d}2^n\right|$$

$$CY2=\left|\frac{YG-YO}{d}2^n\right|$$

onde

d = largura/altura da *quadtree* QUAD2
n = nível máximo da *quadtree* (no GDE = 14)
XO = coordenada X da origem da *quadtree* QUAD2
YO = coordenada Y da origem da *quadtree* QUAD2

É buscado, então, o bloco de QUAD2 que contém o pixel *(CX2,CY2)*. As coordenadas do canto superior esquerdo deste bloco, mais a largura do bloco, são convertidos para o sistema de coordenadas de QUAD1, aplicando-se equações análogas às descritas acima, e devolvidos pela rotina. Estes dois dados são então armazenados nos vetores *xaresta* e *yaresta*.

3.3.7.2. Igualdade de *quadtrees*

Foi desenvolvido um algoritmo para comparar duas *quadtrees* de região ou duas *PMR-f quadtrees*. Primeiramente, duas *quadtrees* só são consideradas iguais se ambas representam a mesma região do espaço, isto é, se possuem a mesma origem e a mesma largura. Além disto, se ambas forem *PMR-f quadtrees*, terão que representar os mesmos fragmentos de linha, mesmo que os representem de forma diferente. Se ambas forem *quadtrees* de região então devem possuir o mesmo conteúdo, isto é, para cada bloco de uma das *quadtrees* existirá na outra *quadtree* o mesmo bloco com a mesma cor.

3.3.7.3. Operações geométricas (perímetro e área)

O algoritmo para calcular a área de uma *quadtree* de região é bastante simples. Para cada bloco preto da *quadtree* lido, converte-se a largura do bloco, em pixels, para as dimensões reais. Calcula-se então a área do bloco. A área da *quadtree* será a soma das áreas de todos os blocos pretos.

Para calcular o perímetro de uma *quadtree* de região linear, utilizou-se o algoritmo descrito em [Samet83]. Ele utiliza os vetores *xaresta* e *yaresta* para armazenar a borda ativa da *quadtree*, com isso evita-se o cálculo de vizinhos. Estes vetores armazenam a largura e a cor dos blocos ativos.

O cálculo do perímetro é realizado pelo GDE através das rotinas INCREMENTA e PERIMETRO. A rotina PERIMETRO inicializa a posição 0 dos vetores *xaresta* e *yaresta* com a largura 2^n e com a cor BRANCA, e para cada bloco da *quadtree* ela chama a rotina INCREMENTA duas vezes: uma para o lado superior do bloco e outra para o lado esquerdo do bloco. O perímetro da *quadtree* é a soma de todos os retornos da rotina INCREMENTA, convertido para o sistema de coordenada global da *quadtree*.

A rotina INCREMENTA retorna, em pixels, quanto o lado superior ou o lado esquerdo do bloco que esta sendo processado contribui para o aumento do perímetro. Se for passado como parâmetro o vetor *xaresta*, ela calculará a contribuição do lado superior do bloco, opostamente, se o parâmetro for o vetor *yaresta* ela calculará a contribuição do lado esquerdo do bloco. Isto acontece porque, como os blocos da *quadtree* são processados pela ordem de Morton, quando um bloco *B2*, com largura *L* e cor *C* posicionado em *(cx,cy)*, for processado os blocos que estiverem localizados acima dele e do seu lado esquerdo já terão sido processados.

Seja *B1* um vizinho ao norte de *B2*, então todos os seus dados relevantes, isto é, a sua largura e a sua cor, se encontram em *xaresta[cx]*. Então existirão três possibilidades:

1) *B1* e *B2* são do mesmo tamanho, então o bloco *B1* corresponde a um bloco folha. Resta verificar se a aresta entre os blocos é CINZA, isto é, se os dois blocos possuem cores diferentes. Caso afirmativo, o perímetro é aumentado de L, isto é, a rotina INCREMENTA irá retornar o valor L. Em qualquer um dos casos, o vetor *xaresta[cx]* é atualizado para descrever a nova borda ativa.
   Por exemplo, quando se processa o bloco 3 da figura 24 posicionado em (0,2), *xaresta[0]* armazena a descrição do bloco 2 (isto é, *xaresta[0].largura* = 2 e *xaresta[0].cor* = BRANCO) . O tamanho do bloco 1 é igual ao tamanho do bloco 3. Como a aresta entre os dois blocos possui a cor BRANCA, pois a cor de ambos os blocos é BRANCA não altera-se o valor do perímetro. Neste caso, não será necessário nenhum tipo de atualização sobre *xaresta* pois o bloco 1 e 3 possuem o mesma cor e o mesmo tamanho.

2) *B1* é maior que *B2*, então *B1* tem que corresponder a um bloco folha. Se a aresta entre *B1* e *B2* for CINZA, o perímetro será aumentado de *L*. Por exemplo, ao se processar o bloco 4 da figura 24, posicionado em (2,2), *xaresta[2]* armazena a cor e o tamanho do bloco 2, ou seja, (cor = BRANCO, largura = 4). Como a aresta entre o bloco 2 e o bloco 4 é BRANCA o perímetro não é alterado. O campo COR de *xaresta[2]* continuará contendo a cor BRANCA, mas o campo LARGURA será alterado de 2 para 1. Como o bloco 2 é maior que o bloco 4, será formada uma nova aresta ativa em *xaresta[3]*, relativa a parte do bloco 2 que não foi processada. Portanto *xaresta[3].largura* será igual a dois e *xaresta[3].cor* será igual a BRANCO.

3) *B1* corresponde a um bloco CINZA, pois *xaresta[cx].largura* < *largura*. São processados todos os blocos entre *B2* e o descendente de *B1* que se encontra mais ao norte. Para cada aresta CINZA, o perímetro é aumentado da largura da aresta.
   Por exemplo, quando processamos o bloco 9 da figura 24 posicionado em *(0,4)*, temos que a largura do bloco 9 é quatro, enquanto o valor de *xaresta[0].largura* é dois. Como a aresta entre os blocos 3 e 9 é CINZA o perímetro é incrementado de *L*=2 (largura do bloco 3). A seguir é processado o bloco 6 que é referenciado em *xaresta[cx+L=2]*. Como a aresta entre eles também é cinza o perímetro é somado um ao perímetro. Por último, proces-

sa-se o bloco 7, que por possuir a mesma cor que o bloco 9 não contribui para o aumento do perímetro. Após processar todos os blocos que se encontram ao norte do bloco 9 é então atualizado o vetor *xaresta*. A posição 0 possuirá cor PRETA (a mesma do bloco 9) e tamanho igual a quatro.

Quando se processa um bloco que é adjacente a uma das bordas da imagem, a rotina determina se a aresta formada pelo bloco e a borda é CINZA (é assumido que a imagem é circundada pela cor BRANCA). Se o for, o perímetro é aumentado.

Para calcular a contribuição do lado esquerdo de um bloco ao perímetro aplica-se o mesmo raciocínio utilizado no processamento do vetor *yaresta*.

A rotina para calcular o perímetro de uma *PMR-f quadtree* é também muito simples. A rotina apenas soma o comprimento de todos os fragmentos representados pela *quadtree*. Para tanto esta realiza, para cada bloco da *quadtree*, um recorte [Newma78] do quadrado do bloco com cada um dos segmentos representados. Os comprimento dos fragmentos obtidos após o recorte são então somados.

3.4. Processador de Consultas e Exibição - PCE

O PCE é responsável pela união do GDE com o COPPEREL; recebe as consultas do usuário em LOPEREL*, através de janelas gráficas, e as processa.

O COPPEREL não reconhece diretamente os dados espaciais, já que neste não existem os tipos ponto, linha ou região. É o PCE que reconhece tais tipos e os interpreta, resolvendo o que é espacial, e mandando o COPPEREL realizar somente as partes relacionais dos comandos. Com o desenvolvimento do PCE evitou-se alterar o código do COPPEREL, de forma a fazê-lo reconhecer diretamente os tipos espaciais, e chamar as funções do GDE quando necessárias. Tal alteração seria mais difícil e trabalhosa do que criar o PCE. O PCE é responsável por interpretar e executar os comandos fornecidos pelo usuário, utilizando tanto o COPPEREL quanto o GDE.

Visando a minimizar o tempo de execução do GEOCOPPE, na implementação dos comandos copiar, selecionar e juntar (da LOPEREL*), o PCE chama diretamente rotinas do COPPEREL, ao invés de fazer chamar o COPPEREL através da interface imbutida. Se tal não fosse feito o tempo de execução seria praticamente inviável. O código de algumas rotinas do COPPEREL foi alterado de modo a incluir chamadas diretas do PCE ao GDE. Isto faz com que aparentemente nessas rotinas o COPPEREL "veja" os dados espaciais.

No entanto, isto não acontece, pois através de chamadas convencionais o COPPEREL não consegue acessar as partes das rotinas que foram alteradas. Tais partes só são acessáveis pelo COPPEREL via PCE, ou seja, quando uma tabela é espacial e quando o PCE manda o COPPEREL manipulá-la. O COPPEREL não pode acessar diretamente estas partes das rotinas, devido à existência nelas das chamadas ao GDE. Como as rotinas do GDE, por sua vez, fazem chamadas ao COPPEREL, ocorreriam dois níveis de chamadas. Para que o COPPEREL realmente "visse" os dados espaciais, além de alterar seu compilador, que precisaria entender a LOPEREL*, teriam que ser substituídas no GDE as chamadas ao COPPEREL por chamadas à máquina virtual. Tais alterações transcendem aos objetivos deste trabalho.

3.4.1. Esquema espacial do GEOCOPPE

O PCE cataloga em REL_ESPACIAIS o nome de todas as relações que possuam algum atributo espacial, e em ATR_ESPACIAIS as informações sobre os atributos espaciais. Os atributos destas duas relações são mostrados nas tabelas 5 e 6:

REL_ESPACIAIS			
Nome	**Tipo**	**Chave**	**Descrição**
NOME_RELACAO	caracter	sim	nome da relação
GRAU	inteiro	não	número de atributos espaciais
TAMANHO_TUPLA	inteiro	não	tamanho da tupla, em caracter, que é devolvida pelo comando mostrar do COPPEREL

Tabela 5 - Descrição da relação REL_ESPACIAIS do esquema espacial

ATR_ESPACIAIS			
Nome	**Tipo**	**Chave**	**Descrição**
NOME_RELACAO	caracter	sim	nome da relação
NOME_ATRIBUTO	caracter	sim	nome do atributo
TIPO_ATRIBUTO	inteiro	não	tipo do atributo espacial: 1 = PONTO 2 = LINHA 3 = REGIÃO
POSICAO_RELATIVA	inteiro	não	posição relativa do atributo (igual a existente em tab_entidades)
POSICAO_REGISTRO	inteiro	não	posição (em palavras) do atributo dentro do registro (igual a existente em tab_entidades)
EST_INDICE	inteiro sem sinal	não	ponteiro para a estrutura de índice aplicada sobre o atributo espacial
POSICAO_SAIDA	inteiro	não	posição do início do atributo na tupla devolvida pela interface do COPPEREL.

Tabela 6 - Descrição da relação ATR_ESPACIAIS do esquema espacial

O atributo EST_INDICE só é utilizado para o tipo ponto, pois a *PR quadtree* funciona como um indexador sobre este tipo. O valor do atributo EST_INDICE é usado para calcular o nome da *PR-quadtree* que armazena os pontos referentes a um atributo *X* de uma relação *Y*. De modo a agilizar alguns tipos de consulta, recomenda-se implementar uma estrutura de índice sobre os atributos do tipo linha e região. A árvore R [Rouss85] [Selli87] [Guttm84] é um exemplo de uma estrutura que funciona como índice sobre as *quadtrees* de região e as *PMR-f quadtrees*. De modo geral, qualquer estrutura que indexe uma coleção de retângulos pode ser usada como índice sobre estes dois tipos de *quadtrees*. Tal implementação está fora do escopo desta tese.

3.4.2. Representação dos dados espaciais

O tipo ponto é armazenado no COPPEREL como um inteiro sem sinal. Caso o atributo do tipo ponto não seja especificado pelo usuário como um atributo chave, é criado um índice para torná-lo um atributo indexado. O inteiro sem sinal que representa o tipo ponto é o código gerado e devolvido pelo GDE, ao se incluir um ponto novo. Quando é inserido um ponto numa tupla, o PCE primeiro manda o GDE inserir o ponto na *PR quadtree*, e depois insere na tupla o código do ponto devolvido pelo GDE. Caso o PCE necessite da coordenada de um atributo do tipo ponto, cujo valor armazenado seja *X*, terá que pedir para o GDE devolver a coordenada do ponto cujo código é *X*.

Os tipos linha e região são armazenados no COPPEREL também como números inteiros sem sinal, que significam respectivamente o nome da *PMR-f quadtree* e *quadtree* de região que os representam.

Se duas tuplas possuírem dois atributos do tipo linha ou região iguais, isto é, as *PMR-f quadtree* ou *quadtrees* de região utilizadas para representá-los são consideradas iguais, existirão duas *quadtrees*, uma para cada tupla. Deste modo, cada atributo do tipo linha ou região possui uma *quadtree*, sendo o PCE responsável pelo gerenciamento dessas *quadtrees*. Se for removida uma tupla que possua um atributo do tipo linha ou região, o PCE mandará o GDE remover a *quadtree* que a tupla referencia. De forma similar, se for removida uma tupla de uma relação que possua um atributo do tipo ponto, o PCE antes de removê-la, recupera o código do ponto e manda o GDE remover da *PR quadtree*, utilizada para armazenar os valores do atributo o ponto, que possui o código recuperado.

3.4.3. Entrada dos comandos da LOPEREL*

O usuário pode entrar com um comando da LOPEREL* apenas por uma janela própria ao comando, por exemplo a janela do comando autorizar usuario mostrada na figura 25. Com exceção dos comandos definir vista, definir asserção e catalogar procedimento todos os outros possuem uma janela própria. Já os comandos da LOPEREL (subconjunto da LOPEREL*) podem ser entrados por uma janela própria para o comando, ou por outra janela cujo nome é Comandos Copperel mostrada na figura 1. Esta janela possui apenas um campo de texto para que o usuário digite integralmente o seu comando. Os comandos fornecidos pela janela Comandos Copperel não podem manipular relações espaciais. Tal janela permite a entrada de qualquer texto, e o envia diretamente para o

COPPEREL sem que antes seja feito qualquer tipo de análise (análise sintática e semântica), enquanto que nas janelas próprias, o PCE gerencia a execução do comando, enviando-o ao COPPEREL se for o caso.

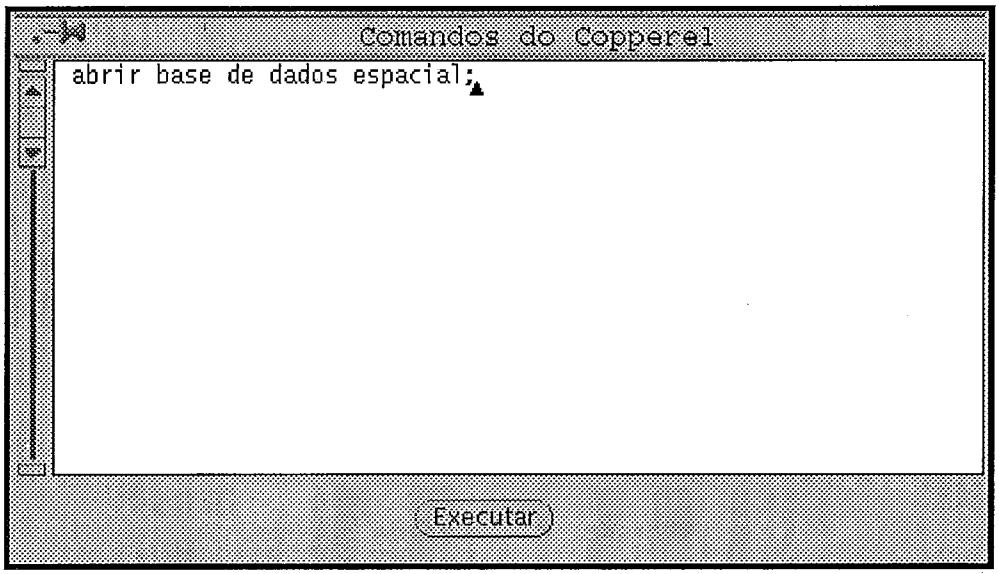


Figura 1 - Tela comandos copperel do PCE, usada para a entrada direta dos comandos LOPEREL

Os comandos: definir vista, definir asserção e catalogar procedimento não são interpretados pelo PCE. Quando utilizados, o PCE simplesmente abre a janela Comandos Copperel. Caso o usuário envie pela janela Comandos Copperel comandos que utilizem relações espaciais, o COPPEREL retornará mensagem de erro.

Conseqüentemente, não funcionam no GEOCOPPE vistas, asserções e procedimentos que manipulem relações espaciais, pois eles não são analisados pelo PCE. Na atual implementação não é possível fazer com que o PCE lide com esses comandos, pois os mesmos são compilados diretamente pelo COPPEREL, que armazena apenas os tokens gerados. No COPPEREL quando o usuário pede para se executar, por exemplo, um procedimento, os tokens previamente armazenados são recuperados e executados. Não há um compilador para a LOPEREL* pois toda a entrada de dados é feita através de janelas. Não é possível gerar tokens para os comandos espaciais, que só existem na LOPEREL*, pois estes não são compilados pelo COPPEREL.

Ainda que a sintáxe das duas linguagens fossem idênticas, o GEOCOPPE, na atual arquitetura, não permitiria que existissem vistas, procedimentos e asserções sobre os dados espaciais. Como o COPPEREL não reconhece os dados espaciais a base poderia tornar-se inconsistente. Por exemplo, caso dentro de um procedimento existisse um comando remover sobre uma relação que possuísse

um atributo do tipo região, o COPPEREL simplesmente removeria as tuplas da relação, sem remover as *quadtrees* de região que as tuplas referenciam.

Quando é pedido para o PCE executar um comando, é verificado se há alguma relação espacial, por consulta à REL_ESPACIAIS através do seguinte comando, enviado ao COPPEREL:

selelcionar *rel_espaciais*
tal que *nome_relação=<nome das relações>* em *$aux1;*.

Se a cardinalidade de *aux1* for igual a zero, significa que não está envolvida no comando nenhuma relação espacial. Neste caso, o PCE enviará o comando diretamente para o COPPEREL, e se for necessário apresenta os resultados obtidos pelo COPPEREL para o usuário (por exemplo, comando mostrar). Caso exista atributos espaciais na relação, o PCE terá que processar parte do comando.

3.4.4. Implementação dos Comandos da LOPEREL*

Foram implementados todos os comandos de sessão, gerência, entrada e saída, e três comandos de manipulação (selecionar, juntar, copiar).

Nos comandos onde é necessário fornecer o nome de arquivos/vistas/asserções/procedimentos/variáveis já existentes dentro da base de dados, o PCE recupera o nome dos arquivos/vistas/asserções/procedimentos/variáveis possíveis de serem escolhidos e apresenta-os ao usuário sob a forma de uma lista. Por exemplo, caso o usuário queira renomear uma tabela, o PCE listará em uma caixa de rolamento todas as tabelas que o usuário está autorizado a escrever.

Após preencher todos os campos necessários para a execução de um comando, o usuário aciona o botão com o rótulo executar.

A seguir é apresentada a metalinguagem adotada para descrever os comandos da LOPEREL* que se seguem. Só serão abordados os comandos que foram de alguma forma afetados pela introdução dos dados espaciais no GEOCOPPE.

Metalinguagem

As regras sintáticas da linguagem são denotadas utilizando-se os seguintes metasimbolos:

Não-terminais - símbolos apresentadados em letras itálicas separadas ou não pelo hífen (-).

Terminais - símbolos apresentados em letras normais (incluindo ou não o sublinha (_)), digitos decimais e os seguintes caracteres: " : ( ) # ## , . $ @ + - * / = <> < <= > >= [ ] ;

Notação

a) O elemento sintático entre colchetes [ ] é opcional.

b) Das opções entre chaves { }, um dos elementos sintáticos deve ser necessariamente escolhido.

c) Quando existirem três pontos ..., significa que o elemento sintático anterior poderá ser escrito uma ou mais vezes.

d) Quando exitirem três caracteres consecutivos sublinhados (ex. ...), significa que o elemento sintático anterior poderá ser escrito uma ou mais vezes com o caracter sublinhado como separador entre eles (no caso o ponto).

e) As constantes espaciais são representadas do seguinte modo:

(x,y) - ponto

(x1,y1);(x2,y2) - segmento de linha

(x1,y1;x2,y2) - região

3.4.4.1. Comando relacional COPIAR envolvento relação espacial

Tal comando em LOPEREL* pode ser escrito do mesmo modo que na linguagem LOPEREL:

copiar [: *atributo1* para *atributo2* ... de] *arquivo1* em [$]*arquivo2*

Na cópia de uma relação *A* que possua atributos espaciais para uma relação *B*, não é suficiente simplesmente o PCE mandar o COPPEREL executar o comando copiar *A* para *B*. Será necessário também incluir a relação *B*, caso ela não exista, no esquema espacial (REL_ESPACIAIS e ATR_ESPACIAIS). Além disto, se a relação *A* possuir um atributo do tipo ponto, terá que ser feita uma cópia da *quadtree* que "indexa" os pontos (valores) deste atributo, e colocar em ATR_ESPACIAIS o seu nome. Só então o PCE manda o comando copiar para o COPPEREL. Se a relação possuir algum atributo do tipo linha ou região, o PCE também não poderá mandar o COPPEREL executar diretamente o comando copiar, pois para cada tupla de *A* será necessário fazer uma cópia das *quadtrees* que representam as linhas ou regiões, e só então inserir a tupla em *B*, alterando antes o nome das linhas ou regiões existentes na tupla para o nome da *quadtree* criada (cópia).

Como exemplo, é apresentada a seguinte relação *A*, que possui três tuplas. O atributo *atributo1* é do tipo inteiro, e o atributo *atributo2* é do tipo região.

ATRIBUTO1	ATRIBUTO2
1	0000000001
2	0000000002
4	0000000003

Para realizar a cópia de *A* para *B*, o PCE faz com que o GDE crie três novas *quadtrees* digamos Q0000000006, Q0000000007, Q0000000008. Sendo que Q0000000006 é cópia de Q0000000001, Q0000000007 é cópia de Q0000000002 e Q0000000008 é cópia de Q0000000003. A relação *B* então conterá as seguintes tuplas.

ATRIBUTO1	ATRIBUTO2
1	0000000006
2	0000000007
4	0000000008

A relação *A* é semanticamente igual à relação *B*, no entanto, caso se fizesse uma união entre as duas relações diretamente pelo COPPEREL, obter-se-ia uma relação com seis tuplas. Isto porque seriam comparados os nomes das *quadtrees*, e não seus conteúdos, isto é, as regiões.

A princípio, pode-se pensar que dividindo o nome da *quadtree* em duas partes, de modo que a primeira especifique a relação espacial a qual pertence, e a segunda especifique o seu nome, não seria preciso mais copiar a relação *A* tupla a tupla para a relação *B*. (A primeira parte seria um atributo da relação não sendo armazenada junto com a segunda parte, mas sim nas tuplas). No entanto, ainda seria necessário recuperar uma a uma as tuplas da relação *A*, para copiar as *quadtrees* referenciadas para as novas *quadtrees* que a relação *B* referenciará.
Na figura 2 o usuário esta realizando a cópia da relação *medicao* para a relação temporária *tab_aux1*. Na figura 3 o usuário esta copiando para a relação *contem2* apenas alguns atributos do arquivo *contem* ("copiar correspondendo").

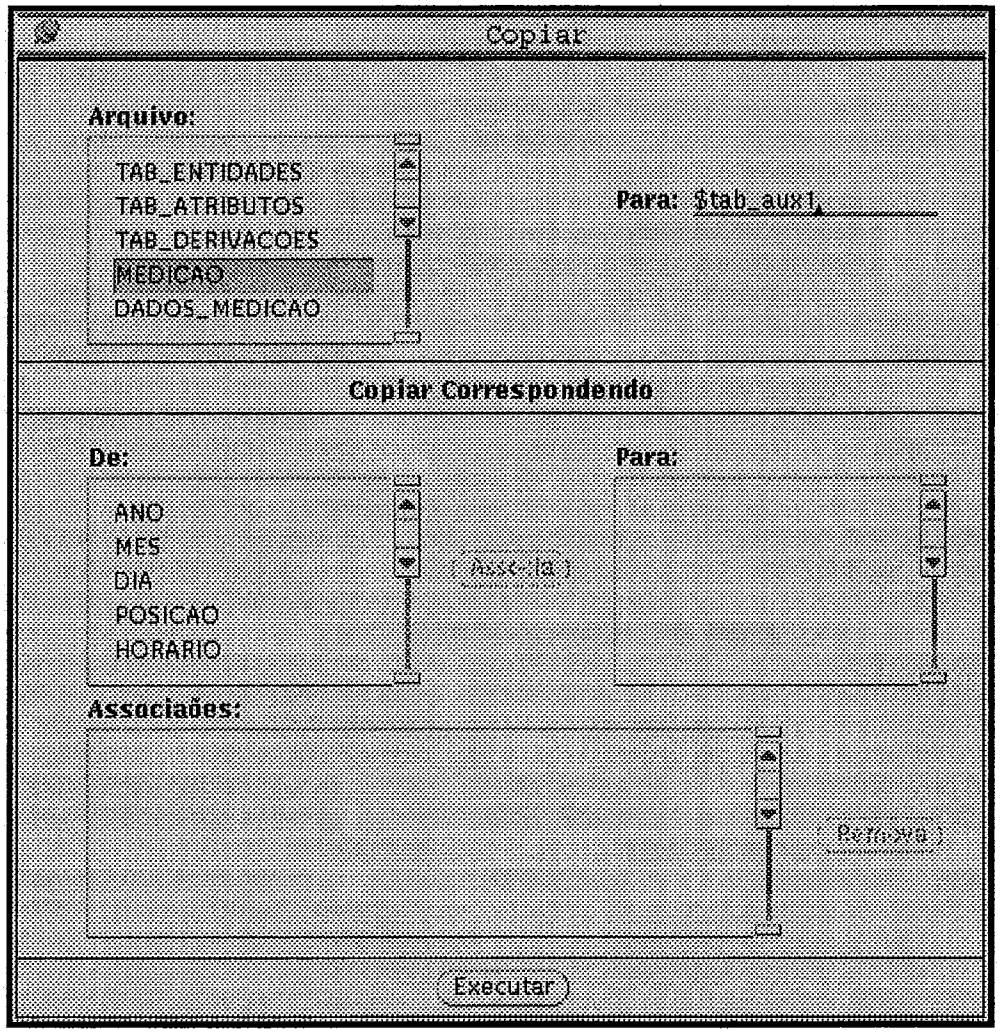


Figura 26 - Janela do comando copiar do PCE.

Implementação

Para implementar o comando copiar foi alterado diretamente o código das rotinas COPIAR, VRUNCL e CPMASS do COPPEREL. A rotina COPIAR realiza a cópia de um arquivo para outro, a rotina VRUNCL verifica se dois arquivos são união compatíveis e a rotina CPMASS copia os registros de uma relação para outra, inserindo nos índices.

A rotina COPIAR realiza três tipos de cópia: a cópia de registros selecionados anteriormente de uma relação para outra; o copiar correspondendo; e a cópia de todos os registros de uma relação para outra. Na execução do comando copiar só são utilizados os dois últimos tipos de cópia. No entanto, o primeiro tipo também foi modificado por ser necessário na implementação do comando selecionar, quando este utiliza relações espaciais.

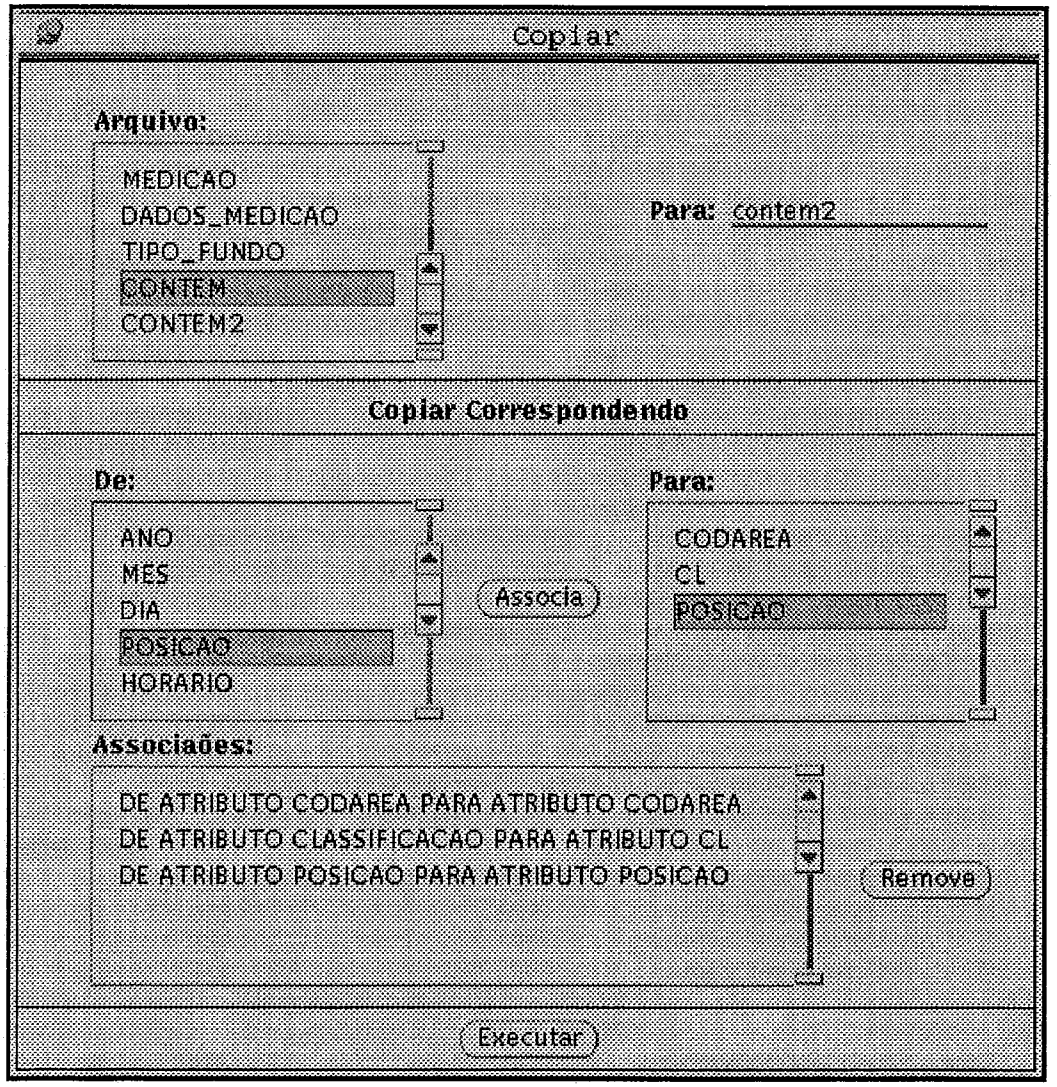


Figura 27 - Janela do comando copiar do PCE, onde esta sendo requisitado o "copiar correspondendo".

A rotina CPMASS é chamada pela rotina COPIAR, quando esta realiza o terceiro tipo de cópia. Originalmente, a rotina apenas lia os registros da relação *arquivo1* e os escrevia na relação *arquivo2*; agora CPMASS faz o seguinte:

Algoritmo da rotina CPMASS

É passado para CPMASS a estrutura *geocoppe*. O campo chamado *viaexp* desta estrutura contém uma lista informando qual a posição e o tipo dos atributos espaciais existentes na relação *arquivo1*.

SE a relação *arquivo1* for espacial (Obs.: esta informação é passada para a rotina através da estrutura *geocoppe*)
PARA *i=1* até o último registro de *arquivo1* faça:
Lê o registro número *i* de *arquivo1* para uma área de memória.
PARA *j=1* até número de atributos espaciais (*geocoppe.viaexp[0]*).
SE o tipo do atributo espacial não for *ponto*
Recupera da área de memória, o nome da *quadtree* que representa o atributo *j*, digamos *quad1*.

Cria uma nova *quadtree*, digamos *quad2*, que é igual a *quad1*.
Escreve na área de memória o nome de *quad2* no lugar de *quad1*.
FIMSE
FIM
Escreve a área de memória na relação *arquivo2*
FIM
FIMSE
SE a relação *arquivo1* não for espacial
Realiza o procedimento de cópia de registros que era feito anteriormente.
FIMSE

Algoritmo da rotina ESQUEMA_ESPACIAL

Esta rotina é chamada pela rotina COPIAR, quando for pedido um dos seguintes tipos de cópias: o **copiar registros selecionados** ou o **copiar todos os registros.** É passado como parâmetro para a rotina uma variável que especifica o seu contexto de execução, isto é, qual o tipo de copia que esta sendo relalizado.

O objetivo da rotina é colocar em *geocoppe.viaexp* informações sobre os tipos espaciais, que são retiradas do esquema espacial. Se o tipo do atributo for ponto ela armazenará o tipo (ponto), a posição dentro do registro, o nome da *quadtree* que armazenava os pontos da relação fonte (*arquivo1*) e o nome da *quadtree* que armazenará os pontos da relação destino (*arquivo2*). Caso contrário, ela apenas armazenará o tipo (linha ou região) e a posição dentro do registro.

Além disto, quando estiver copiando todos os registros de um arquivo, se existir na relação atributos do tipo ponto, ela copiará o conteúdo das *PR quadtrees* da relação *arquivo1* para as *PR quadtrees* da relação *arquivo2.*

PARA cada atributo espacial (*atributo1*) da relação *arquivo2*
Recupera de ATR_ESPACIAIS o tipo de *atributo1* e a sua posição dentro do registro, e os coloca em *geocoppe.viaexp.*
SE o tipo de *atributo1* for ponto
Busca em ATR_ESPACIAIS o atributo da relação *arquivo2* que possui a mesma posição relativa que *atributo1*, para poder recuperar a estrutura de índice de *arquivo1* (*ei1*) e de *arquivo2* (*ei2*)
Calcula o nome das *PR quadtrees* *quad1* e *quad2*, que correspondem respectivamente aos números *ei1* e *ei2*
SE o tipo da cópia for *copiar*
Copia o conteúdo da *PR quadtree* *quad1* para a *PR quadtree* *quad2*
SE *quad2* não existia antes da cópia
Atualiza o atributo EST_INDICE da relação ATR_ESPACIAIS
FIMSE
FIMSE
Coloca a parte inteira do nome de *quad1* e *quad2* em *geocoppe.viaexp*
FIMSE
Coloca em *geocoppe.viaexp[0]* o número de atributos espaciais encontrados
FIM

Algoritmo da rotina COPIAR

Quando a rotina COPIAR é chamada, a relação *arquivo2* já foi previamente criada, ou existia anteriormente. Portanto, ela já existe no esquema relacional (TAB_ENTIDADES e TAB_ATRIBUTOS) e no esquema espacial (REL_ESPACIAIS e ATR_ESPACIAIS).

A rotina foi alterada no seu início de forma a realizar o procedimento descrito no item 1 abaixo. As outras alterações são descritas nos itens 2, 3 e 4.

1) Verificação de relação espacial:

SE a relação *arquivo1* é espacial

O campo *relacao_espacial* da estrutura *geocoppe* recebe o valor *1* (=*verdadeiro*), caso contrário receberá o valor *0* (= *falso*)

FIMSE

2) Caso do pedido para copiar todos os registros da relação *arquivo1* para a relação *arquivo2* (terceiro tipo):

SE *geocoppe.relacao_espacial=verdadeiro* (a relação *arquivo1* é espacial)

Chama a rotina ESQUEMA_ESPACIAL(tipo da cópia=*copiar*)

FIMSE

Executa a rotina CPMASS

3) Caso do pedido para "copiar correspondendo". Originalmente neste tipo de cópia a rotina COPIAR recebia como parâmetro de entrada, para cada par de atributo, através do vetor *maqina_1.viaexp*, as seguintes informações:

- posição do atributo no arquivo operando
- tamanho do atributo do arquivo operando (*t1*)
- posição do atributo no arquivo resultado
- tamanho do atributo do arquivo resultado (*t2*)

Na rotina alterada, quando a relação é espacial, são acrescentados às informações acima descritas os seguintes dados:

- tipo do atributo
- estrutura de índice do atributo do arquivo operando (*ei1*)
- estrutura de índice do atributo do arquivo operando (*ei2*)

SE *geocoppe.relacao_espacial=verdadeiro* (a relação *arquivo1* é espacial)

PARA cada par de atributos a ser copiado

SE o tipo dos atributos (operando e resultado) for *ponto*

Obtém o nome das PR *quadtrees* *quad1* e *quad2*, respectivamente através de *ei1* e *ei2* (passados como parâmetros)

Copia o conteúdo de *quad1* para *quad2*

FIMSE

FIM

FIMSE

PARA cada tupla de *arquivo1*

PARA cada par de atributos a ser copiado

SE a relação *arquivo1* for espacial

Recupera os dados de um par de atributos (passados por *maqina_1.viaexp*) pela formatação antiga

SENÃO

Recupera os dados de um par de atributos (passados por *maqina_1.viaexp*) pela formatação nova

FIMSE

SE $t2 \leq t1$

Recupera o atributo do arquivo operando, e o escreve no registro que contém a tupla do arquivo resultado

SE o atributo for do tipo linha ou do tipo região

Obtém o nome da *quadtree*, *quad1*, que representa a linha ou a região. Ela é calculada a partir do atributo do arquivo operando, que foi lido anteriormente.

Copia *quad1* para uma *quadtree* nova *quad2*, e coloca a parte inteira do seu nome no registro que contém a tupla do arquivo resultado.

FIMSE

SENÃO

Recupera o atributo do arquivo operando, e o escreve no registro que contém a tupla do arquivo resultado.

Preenhe o espaço que sobrou, pois o atributo resultado é menor que o atributo operando com brancos.

```
            FIMSE
        FIM
    FIM
    SE o registro não for duplicata
        Insere o registro em arquivo2
    FIMSE
```

Algoritmo do PCE

As rotinas descritas acima foram alteradas para serem chamadas pelo PCE. O algoritmo implementado pelo PCE é apresentado a seguir:

Se a relação *arquivo2* já existir ela é espacialmente desalocada (para saber como se desaloca uma rotina espacial veja o comando desalocar).

```
SE a relação arquivo1 e a relação arquivo2 não forem espaciais
        Envia o comando para o COPPEREL
    SENÃO SE a relação arquivo1 for espacial
        SE a relação arquivo2 for espacial OU temporária
            SE a relação arquivo2 não existir
                Cria a tabela temporária arquivo2
            FIMSE
            SE não for pedido o copiar correspondendo
                Chama a rotina VRUNCL do COPPEREL (testa se as duas tabelas espaci-
                    ais são união compatíveis
                SE as tabelas não forem união compatíveis
                    ERRO - Envia mensagem de erro
                FIMSE
            SENÃO
                Coloca em maqina_1.viaexp os parâmetros da rotina COPIAR
                SE não foi especificado todos os atributo chaves de arquivo2
                    Envia mensagem de erro
            FIMSE
            Chama a rotina COPIAR do COPPEREL
            SE houve erro de execução na rotina COPIAR (maqina_1.errexe <> 0)
                SE a relação resultado for nova, isto é, não existia antes
                    Remove o arquivo espacialmente, isto é, o remove do esquema
                      espacial, e remove todas as quadtrees referenciadas por ele.
                SENÃO
                    Desaloca o arquivo espacialmente
                FIMSE
            FIMSE
        SENÃO SE foi pedido o copiar correspondendo
            Envia o comando para o COPPEREL
        SENÃO
            ERRO - As tabelas não são união compatíveis
        FIMSE
    SENÃO SE a tabela arquivo2 for espacial
        ERRO - As tabelas não são união compatíveis
    FIMSE
FIMSE
```

3.4.4.2. Comando relacional CRIAR BASE DE DADOS

Tal comando na linguagem LOPEREL* pode ser escrito do mesmo modo que na linguagem LOPEREL.

criar base de dados *bd* em *dispositivo1* e *dispositivo2* para *inteiro* registros por arquivo

Este comando é fornecido através da janela apresentada na figura 28. Na figura o usuário esta criando uma base de dados, de nome *espacial,* com 500 registros por arquivo.

Figura 28 - Janela do comando criar base de dados do PCE

A rotina CRIABD do COPPEREL foi modificada de modo a criar as relações utilizadas pelo PCE (esquema espacial), e as relações utilizadas pelo GDE. A rotina CRIABD também inicializa a estrutura *geocoppe.* A estrutura *geocoppe* armazena os endereços físico das relações usadas pelo GDE e pelo PCE, e é utilizada pelas rotinas do COPPEREL que foram alteradas. Ao se alterar diretamente o código do COPPEREL, tornou-se possível ao usuário fornecer o comando criar base de dados através da janela comandos copperel. Pois o próprio COPPEREL cria as relações utilizados pelo GEOCOPPE no gerenciamento dos dados espaciais.

3.4.4.3. Comando relacional ABRIR BASE DE DADOS

Tal comando na linguagem LOPEREL* pode ser escrito do mesmo modo que na linguagem LOPEREL.

abrir base de dados *bd*

Este comando é fornecido através da janela mostrada na figura 29, nela o usuário esta abrindo a base de dados *espacial*.

A rotina ABREBD (do mesmo modo que a rotina CRIABD) do COPPEREL foi modificada de modo a inicializar a estrutura *geocoppe*, após a abertura da base de dados. Deste modo, o usuário pode fornecer o comando abrir base de dados também pela janela comandos copperel.

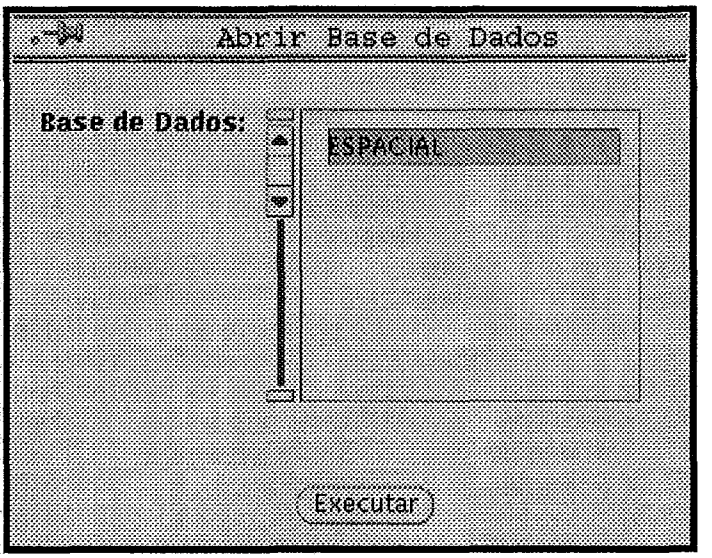


Figura 29 - Janela do comando abrir base de dados do PCE

3.4.4.4. Comandos relacionais DEFINIR VISTA, DEFINIR ASSERCAO e CATALOGAR PROCEDIMENTO

O GDE não possui estes comandos. Caso o usuário os escolha, abrir-se-á a janela de Comandos do Copperel, mostrada na figura 25.

3.4.4.5. Comandos relacionais ABOLIR VISTA, ABOLIR ASSERCAO e ABOLIR PROCEDIMENTO

Tais comandos na linguagem LOPEREL* podem ser escritos do mesmo modo que na linguagem LOPEREL.

abolir vista *vista1*
abolir assercao *asserção1*
abolir procedimento *procedimento1*

Para cada um destes comandos, o PCE abre uma janela (figuras 30 a 32), onde o usuário fornece o nome da vista/asserção/procedimento a ser abolida. Após isto o PCE envia o comando para o COPPEREL.

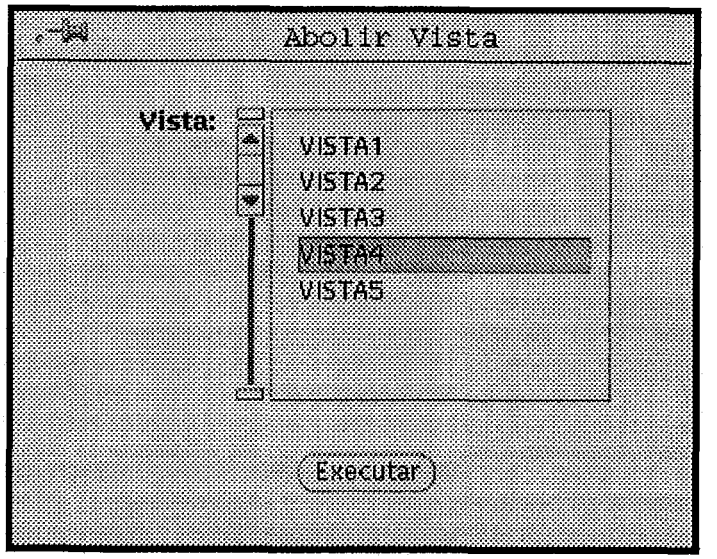


Figura 30 - Janela do comando abolir vista do PCE

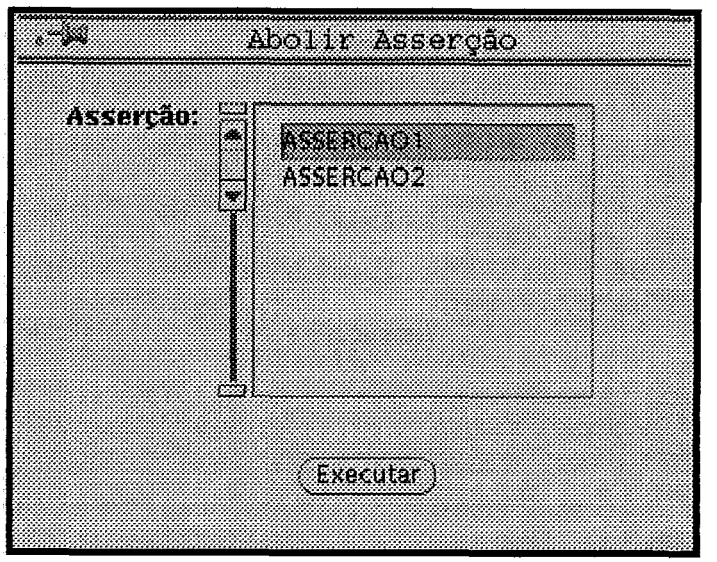


Figura 31 - Janela do comando abolir assercao do PCE

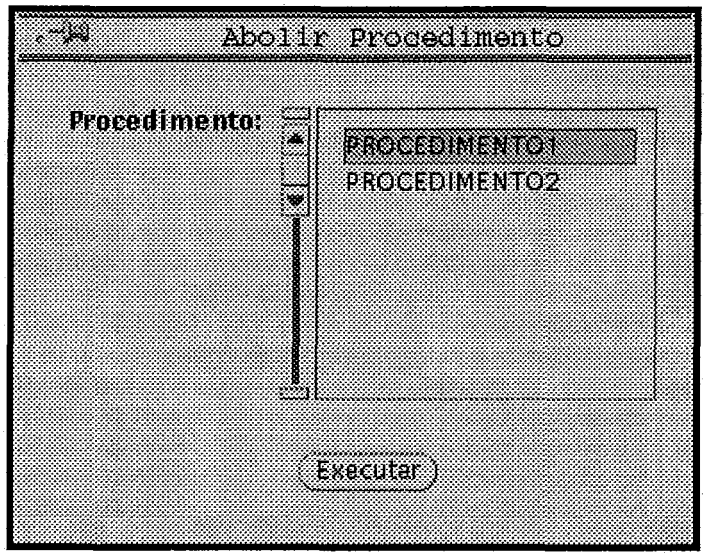


Figura 32 - Janela do comando abolir procedimento do PCE

3.4.4.6. Comando relacional CRIAR VARIAVEL envolvento relação espacial

Este comando na linguagem LOPEREL* pode ser escrito como:

criar variavel *variavel1* { de tipo : } { real (*inteiro:inteiro*) | inteiro (*inteiro*) | intss (*inteiro*) | alfanumerico (*inteiro*) | ponto | linha | região }

As variáveis espaciais são inseridas no esquema espacial. Quando a variável for do tipo ponto, não será criada uma *PR quadtree* associada, pois não há necessidade de uma *PR quadtree* para armazenar apenas um ponto. É criada apenas a relação para armazenar a coordenada do ponto, não sendo inserida nenhuma tupla na relação CODIGO_PONTO. As variáveis do tipo ponto no COPPEREL, tal qual os atributos do tipo ponto das relações, armazenam o código de um ponto, que no caso das variáveis, é sempre igual a 1.

O comando criar variavel é fornecido através da janela mostrada na figura 33, onde se está criando uma variável com o nome *rio_de_janeiro* e do tipo região.

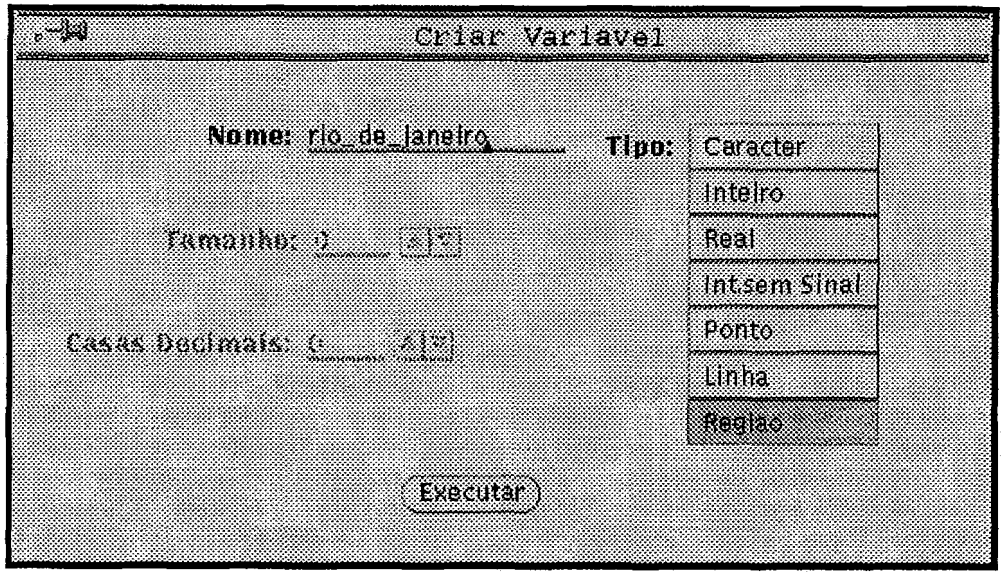


Figura 33 - Janela do comando criar variavel.

Algoritmo

Cria a variável *variavel1* no COPPEREL. Os tipos espaciais serão criados como números inteiros sem sinal
SE não hove erro de execução
    SE *variavel1* for espacial
        SE *variavel1* for do tipo ponto
            Cria a relação que associará a coordenada da variável ao seu código (que será sempre igual a 1). Esta relação possuirá cardinalidade unitária.
        FIMSE
    FIMSE
    Coloca a descrição de *variavel1* em ATR_ESPACIAIS e em REL_ESPACIAIS
FIMSE

3.4.4.7. Comando relacional REMOVER VARIAVEL envolvento relação espacial

Este comando na linguagem LOPEREL* pode ser escrito da mesma forma que na linguagem LOPEREL.

remover variavel *variavel1*

Na figura 34 mostra a tela utilizada para o fornecimento do comando remover variavel.

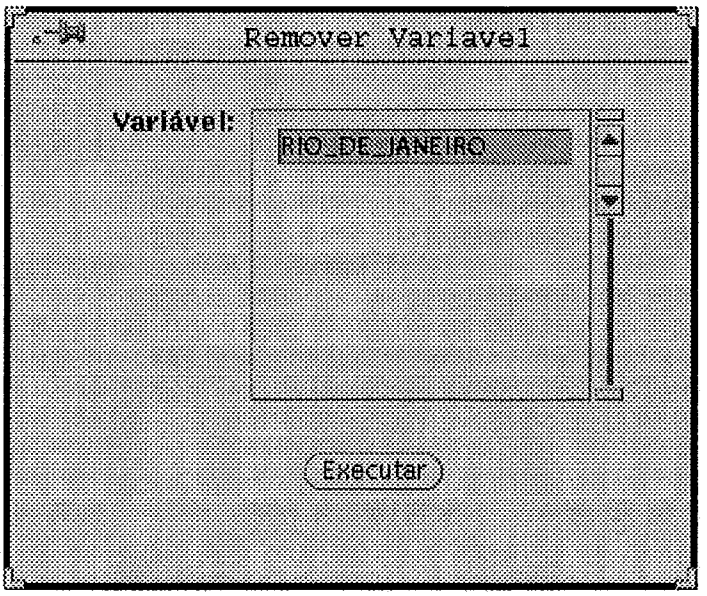


Figura 34 - Janela para a entrada do comando remover variavel.

Se a variável a ser removida for espacial, ela é retirado do esquema espacial. Quando a variável for do tipo linha ou região também é removida a *PMR-f quadtree* e *quadtree* de região associada e, quando a variável for do tipo ponto, é removida a relação que armazena a coordenada do ponto.

Algoritmo

SE *variavel1* for espacial
    SE tipo de *variavel1* for ponto
        Manda o COPPEREL remover o arquivo que contém a coordenada do ponto
    SENÃO
        Remove a *quadtree* que representa a região ou a linha
    FIMSE
    Retira a variável *variavel1* do esquema espacial.
    Envia o comando remover variavel *variavel1* para o COPPEREL
SENÃO
    Envia o comando remover variavel *variavel1* para o COPPEREL
FIMSE
SE ocorreu erro
    Envia mensagem de ERRO
FIMSE

3.4.4.8. Comando relacional REMOVER ARQUIVO envolvento relação espacial

Este comando em LOPEREL* pode ser escrito da mesma forma que na linguagem LOPEREL.

remover arquivo *arquivo1*

Este comando é fornecido pelo usuário através da janela mostrada na figura 35, onde são listados apenas os nomes dos arquivos que o usuário possui permissão para remover.

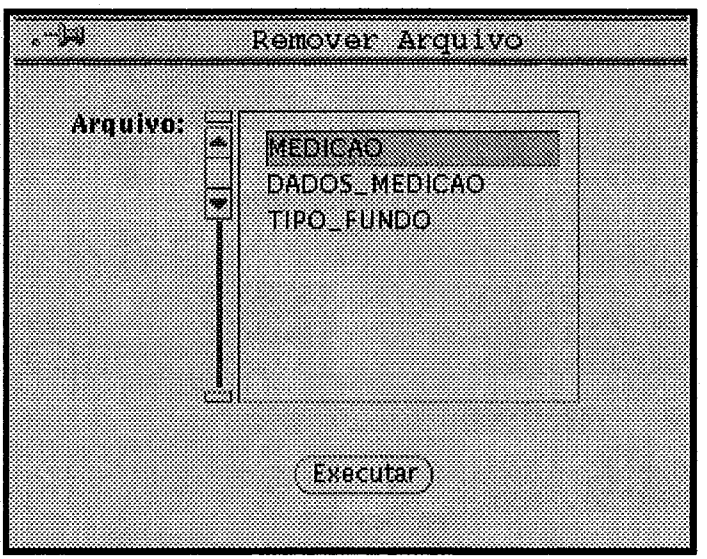


Figura 35 - Janela do comando remover arquivo do PCE.

Ao remover uma tabela espacial, o PCE remove também todas as *quadtrees* que estão associadas a ela, e a retira do esquema espacial. Evitando, assim, que a base de dados se torne inconsistente.

Algoritmo

SE *arquivo1* for espacial
    Remove todas as *quadtrees* de região e *PMR-f quadtrees* associadas a *arquivo1* e retira todos os pontos existentes nas *PR quadtrees*.
    Retira a tabela *arquivo1* do esquema espacial, removendo as *PR quadtrees* existentes.
    Envia o comando remover arquivo *arquivo1* para o COPPEREL
SENÃO
    Envia o comando remover arquivo *arquivo1* para o COPPEREL
FIMSE
SE ocorreu erro
    ERRO - Envia mensagem de erro
FIMSE

Os comandos contidos no primeiro bloco SE/SENÃO formam a rotina chamada REMOVER_ARQUIVO_ESPACIAL.

3.4.4.9. Comando relacional DESALOCAR ARQUIVO envolvento relação espacial

Este comando em LOPEREL* pode ser escrito da mesma forma que na linguagem LOPEREL.

desalocar arquivo *arquivo1*

O comando desalocar arquivo é fornecido pelo usuário através da janela mostrada na figura 36, onde são listados apenas os nomes dos arquivos que o usuário possui permissão para remover.

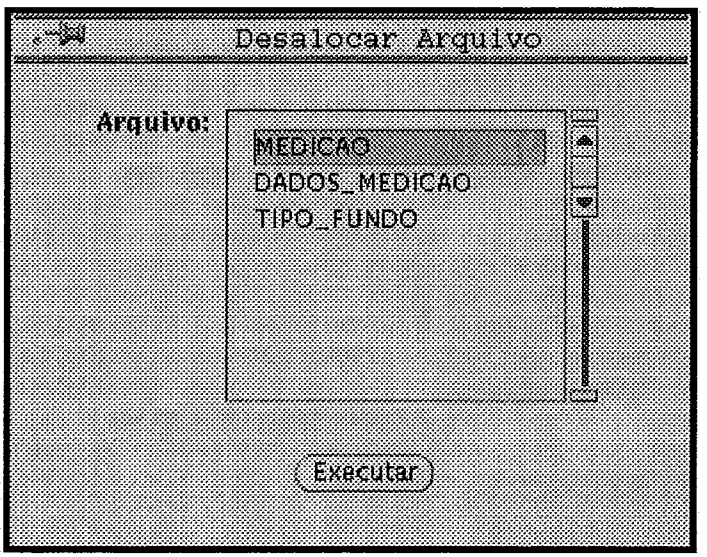


Figura 36 - Janela do comando desalocar arquivo do PCE

Ao desalocar uma tabela espacial, o PCE remove todas as *quadtrees* de região e *PMR-f quadtrees* que estão associadas a ela. As *PR quadtrees* são desalocadas. Diferentemente do comando remover arquivo, a tabela não é removida do esquema espacial.

Algoritmo

```
SE arquivo1 for espacial
    Remove todas as quadtrees de região e PMR-f quadtrees associadas ao arquivo1 e retira
        todos os pontos existentes nas PR quadtrees.
    Envia o comando desalocar arquivo arquivo1 para o COPPEREL
SENÃO
    Envia o comando desalocar arquivo arquivo1 para o COPPEREL
FIMSE
SE ocorreu erro
    Envia mensagem de ERRO
FIMSE
```

Os comandos contidos no primeiro bloco SE/SENÃO formam a rotina chamada DESALOCAR_ARQUIVO_ESPACIAL.

3.4.4.10. Comando relacional RENOMEAR ARQUIVO envolvento relação espacial

Este comando na linguagem LOPEREL* pode ser escrito da mesma forma que na linguagem LOPEREL.

renomear arquivo *arquivo1* para *arquivo2*

Este comando é fornecido através da janela apresentada na figura 37, onde são listados apenas os nomes dos arquivos que o usuário possui permissão para remover. Nesta janela o usuário esta renomeando a tabela *daods_medicao* para *dados_medicao.*

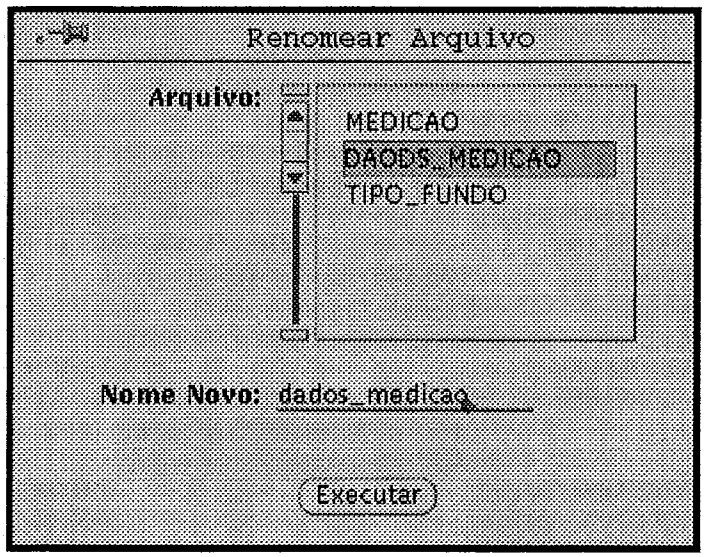


Figura 37 - Janela do comando renomear arquivo do PCE

Algoritmo

Envia para o COPPEREL o comando renomear arquivo *arquivo1* para *arquivo2*
SE não houve erro na execução do comando renomear
    SE *arquivo1* for espacial
        Renomeia *arquivo1* no esquema espacial, i. e., troca o nome da tabela *arquivo1* para *arquivo2* nas tabelas ATR_ESPACIAIS e REL_ESPACIAIS
    FIMSE
SENÃO
    ERRO - Envia a mensagem de erro dada pelo COPPEREL
FIMSE

3.4.4.11. Comando relacional RENOMEAR ATRIBUTOS envolvento relação espacial

Este comando na linguagem LOPEREL* é escrito na mesma forma que na linguagem LOPEREL.

renomear atributos de *arquivo1*: { *atributo1* para *atributo2*}$_{1,1}$

Este comando é fornecido através da janela apresentada na figura 38, onde são listados apenas os nomes dos arquivos que o usuário possui permissão para remover. Nela o usuário especifica a tabela cujos atributos serão renomeados, os atributos a serem renomeados, e os novos nomes. Por exemplo, na figura 38 o usuário esta renomeando o atributo *posicao* da tabela *dados_medicao* para *posicao2*.

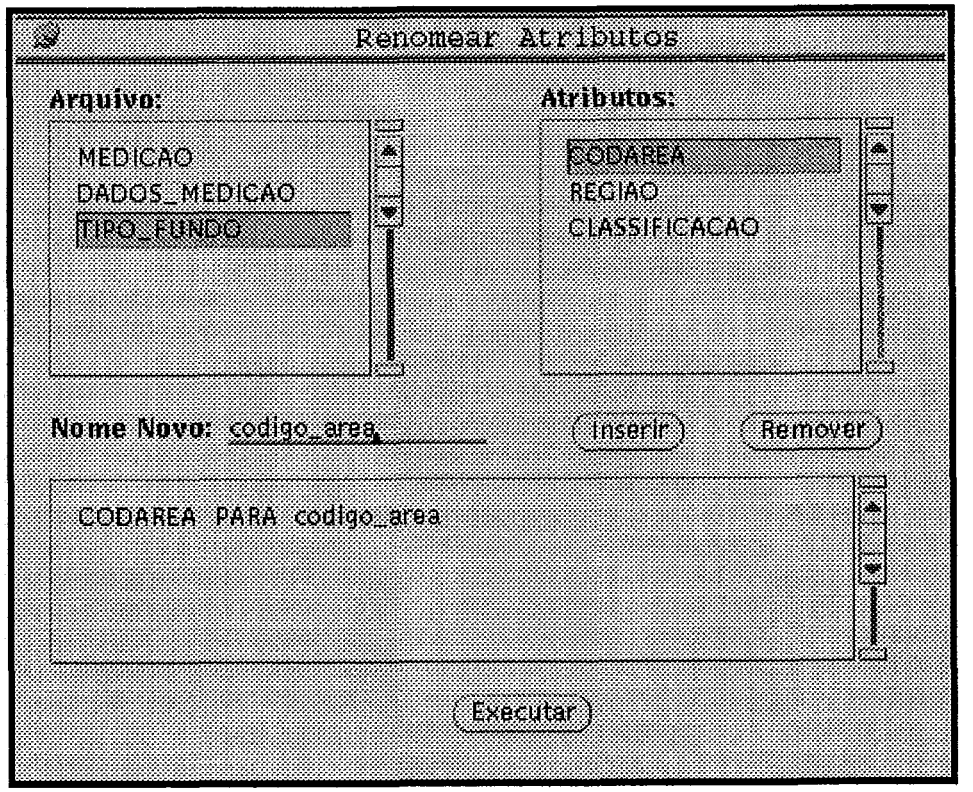


Figura 38 - Janela do comando renomear atributos do PCE

Algoritmo

Envia para o COPPEREL o comando renomear atributos
SE não houve erro na execução do comando renomear atributos
    SE *arquivo1* for espacial
        PARA cada atributo espacial a ser renomeado
            Muda o nome do atributo no esquema espacial, isto é, troca o nome do atributo na tabela ATR_ESPACIAIS.
        FIM
    FIMSE
SENÃO
    ERRO - envia a mensagem de erro dada pelo COPPEREL
FIMSE

3.4.4.12. Comandos relacionais AUTORIZAR USUARIO, REVOGAR AUTORIZACAO, RECONSTRUIR BASE DE DADOS, DESALOCAR VISTA, VERIFICAR INTEGRIDADE e ABANDONAR ORDEM

Estes comandos na linguagem LOPEREL* são escritos da mesma forma que em linguagem LOPEREL.

autorizar usuário *usuario/senha* para { remover | escrever | ler_escrever | ler } *arquivo*

revogar autorizacao de *usuario/senha* para *arquivo*

reconstruir base de dados *bd*

desalocar vista *vista*

verificar integridade com *asserção*

abandonar ordem de *arquivo*

abrir sessão para usuario *usuario/senha*

Para cada um desses comandos existe uma janela própria, que são mostradas a seguir (figuras 39-45).

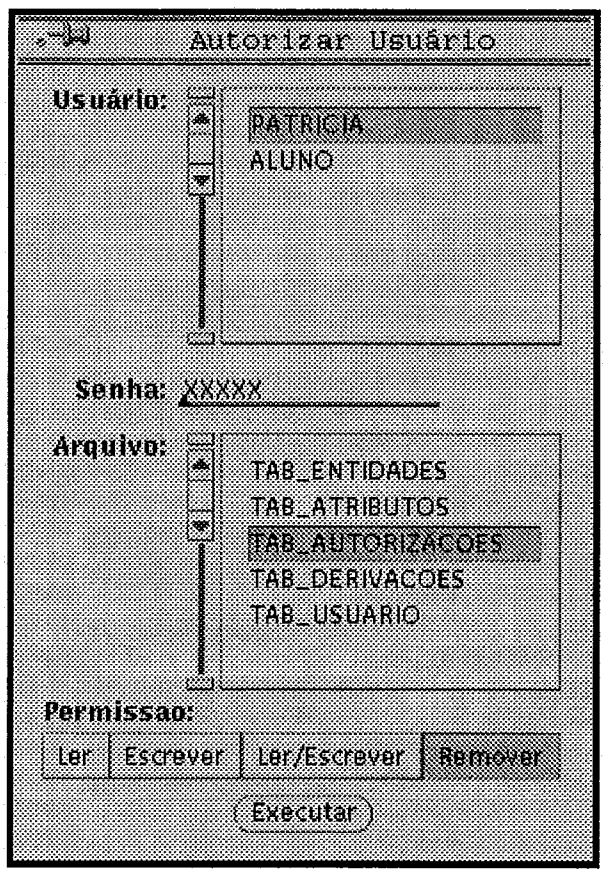


Figura 39 - Janela do comando autorizar usuario do PCE

Através destas janelas, o PCE consegue todas as informações necessárias para a execução do comando, enviando-o a seguir para o COPPEREL. Não será apresentado nenhum algoritmo, pois não é necessário nenhum pré-processamento com os dados espaciais nestes comandos.

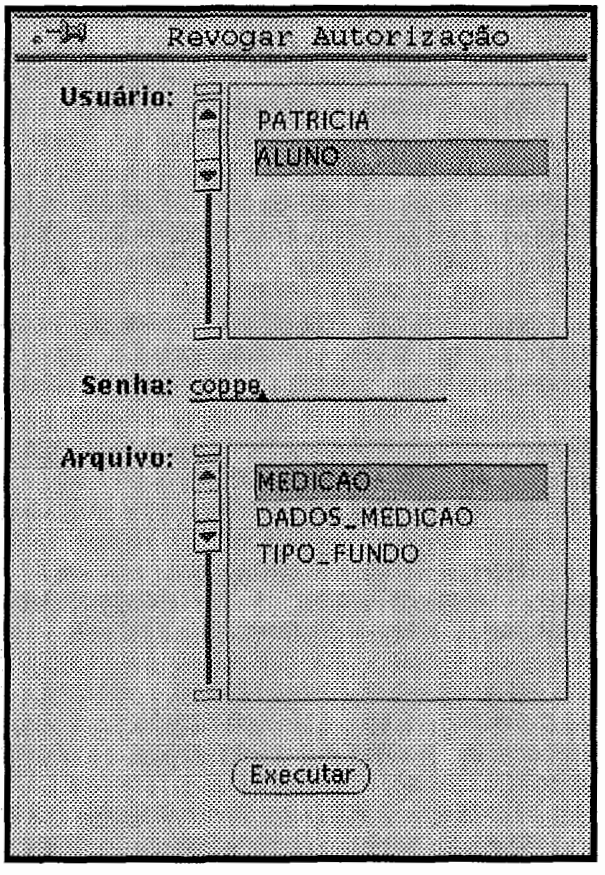


Figura 40 - Janela do comando revogar autorização do PCE

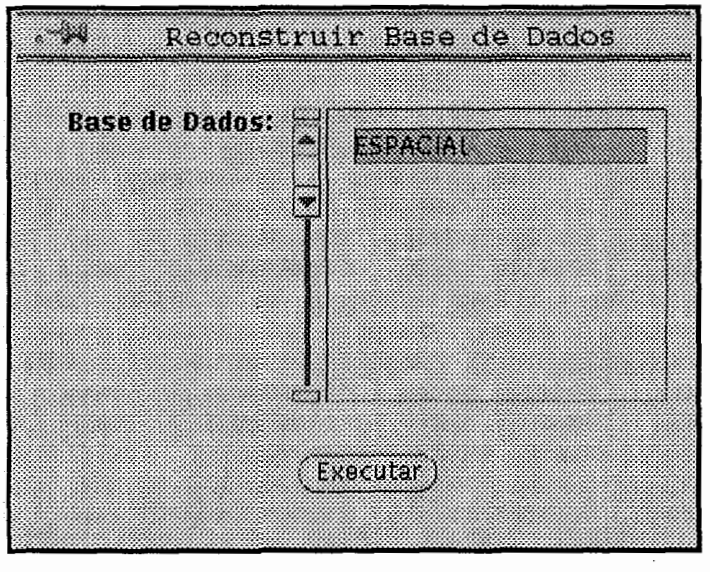


Figura 41 - Janela do Comando reconstruir base de dados do PCE

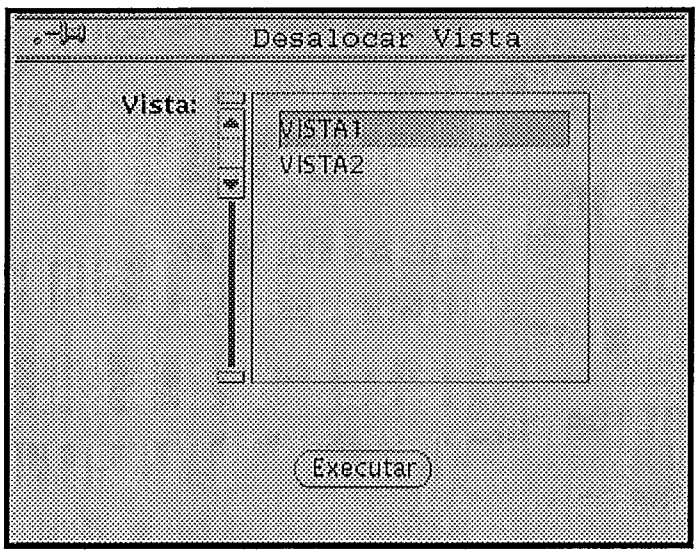


Figura 42- Janela do Comando desalocar vista do PCE

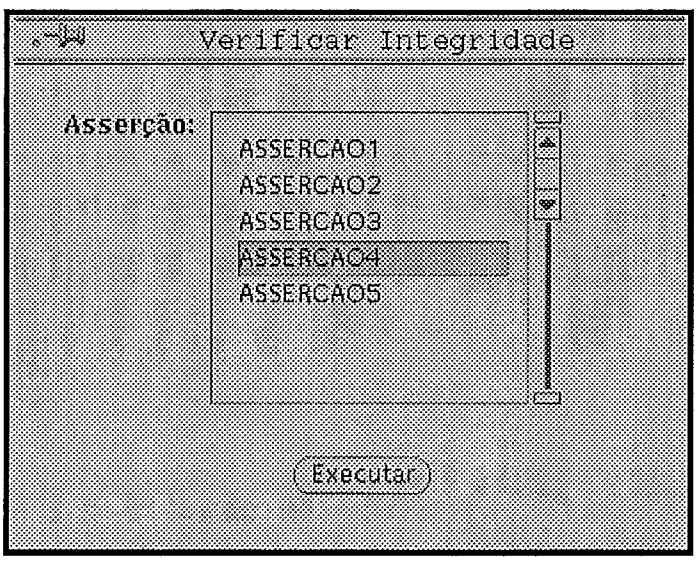


Figura 43 - Janela do Comando verificar integridade do PCE

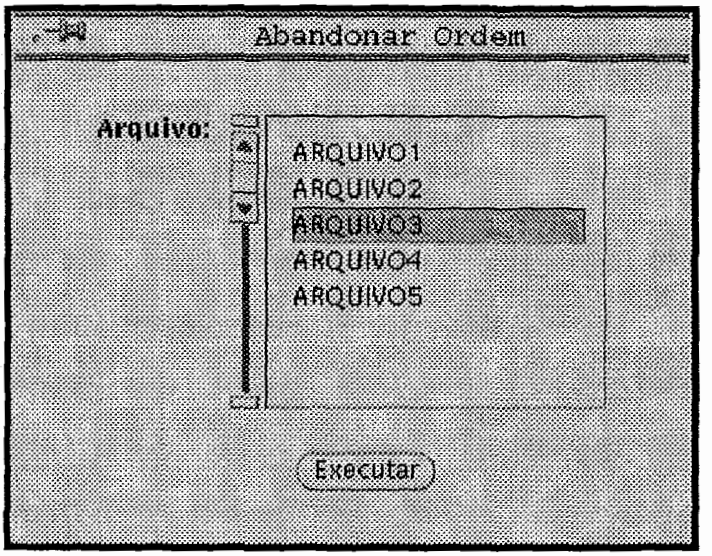


Figura 44 - Janela do comando abandonar ordem do PCE

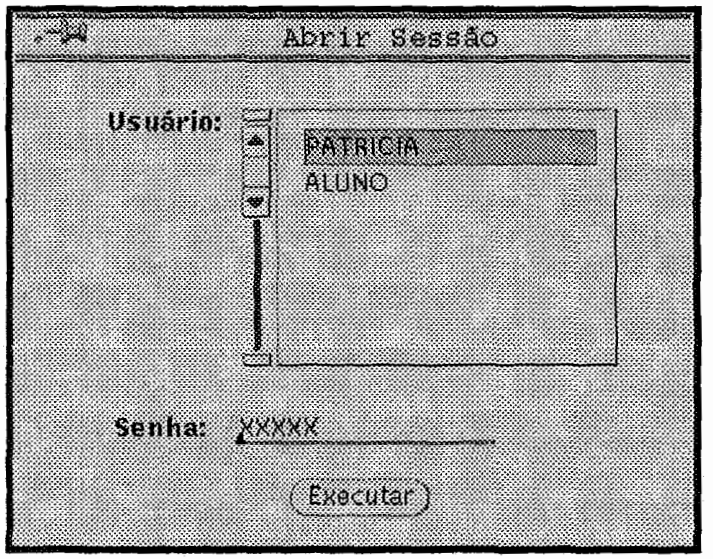


Figura 45 - Janela do comando abrir sessao do PCE

3.4.4.13. Comandos relacionais REMOVER INDICE e CRIAR INDICE

Estes comandos na linguagem LOPEREL* podem ser escritos do mesmo modo que na linguagem LOPEREL.

remover indice de *arquivo* sobre *atributo*$_{,,,}$
criar indice em *arquivo* sobre *atributo*$_{,,,}$

Como já foi mencionado, os atributos espaciais são armazenados no COPPEREL como números do tipo inteiro sem sinal. Desta forma, estes atributos podem ser indexados como qualquer outro tipo relacional pelo COPPEREL. No entanto, em termos espaciais esta indexação não possui sentido, pois não é utilizada pelo GDE em nenhuma operação espacial. O índice que seria criado não possuiria nenhuma característica espacial. Portanto, o GEOCOPPE não permite que o usuário crie índice sobre os atributos espaciais.

O PCE internamente cria índice sobre o atributo espacial pois este é utilizado na otimização de consultas. Em termos espaciais a indexação dos pontos é feita pela *PR quadtree*.

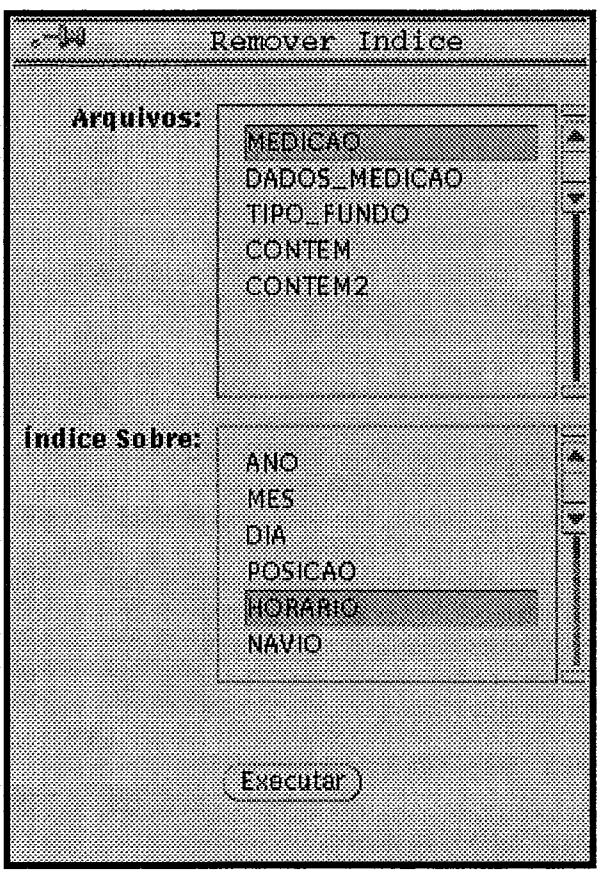


Figura 46 - Janela do comando remover indice do PCE

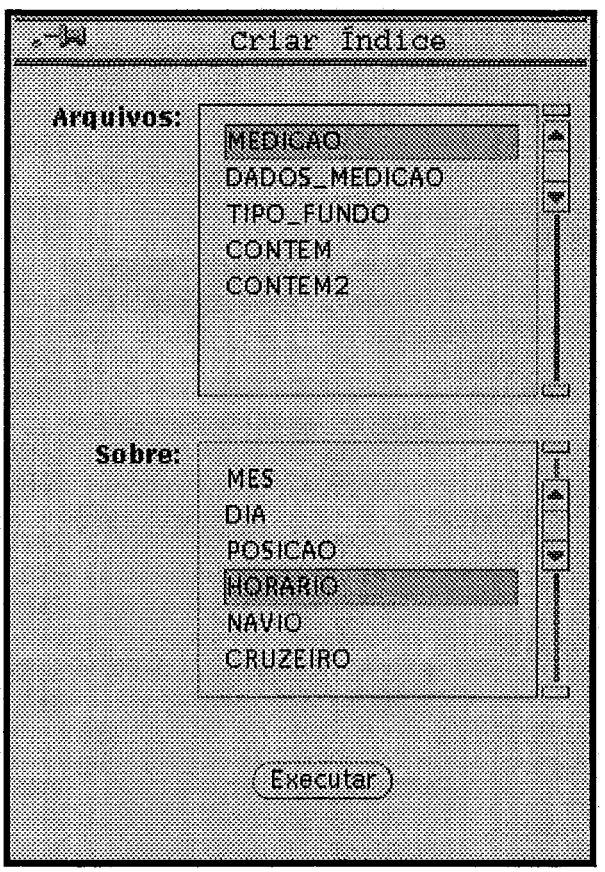
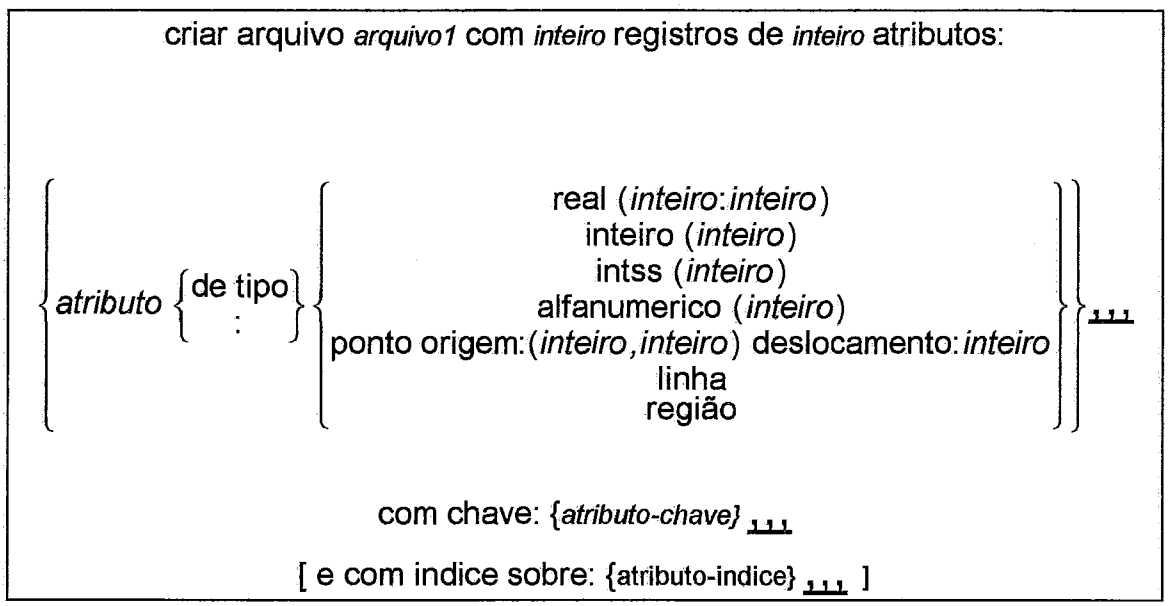


Figura 47 - Janela do comando criar indice do PCE

3.4.4.14. Comando relacional CRIAR ARQUIVO envolvento relação espacial

Este comando na linguagem LOPEREL* pode ser escrito como



Este comando é fornecido através da janela mostrada na figura 48. Nela o usuário fornece o nome da relação a ser criada, o nome e o tipo dos atributos da relação. Para cada atributo do tipo ponto, o usuário fornece a origem (x,y) e a largura/altura (deslocamento) do quadrado que conterá todos os pontos deste atributo.

Os atributos do tipo linha e região não podem ter índice sobre eles. Caso esteja selecionado um atributo de um destes dois tipos, o campo *índice* ou o campo *chave* não estarão ativos. Pois, no tipo *linha* ou *região* o que é armazenado é o nome da *quadtree*, que, apesar de ser também único, não identifica univocamente uma *quadtree*.

De outro modo, é permitido que seja criado um índice sobre um atributo do tipo *ponto*, ou que seja definido como um atributo chave. Pois, o que é armazenado na relação é o código do ponto, que é único, e caso dois pontos sejam iguais seus códigos também o são, caso contrário seus códigos são diferentes.

Para agilizar as operações sobre o tipo ponto ele é sempre criado no COPPEREL como um atributo indexado.

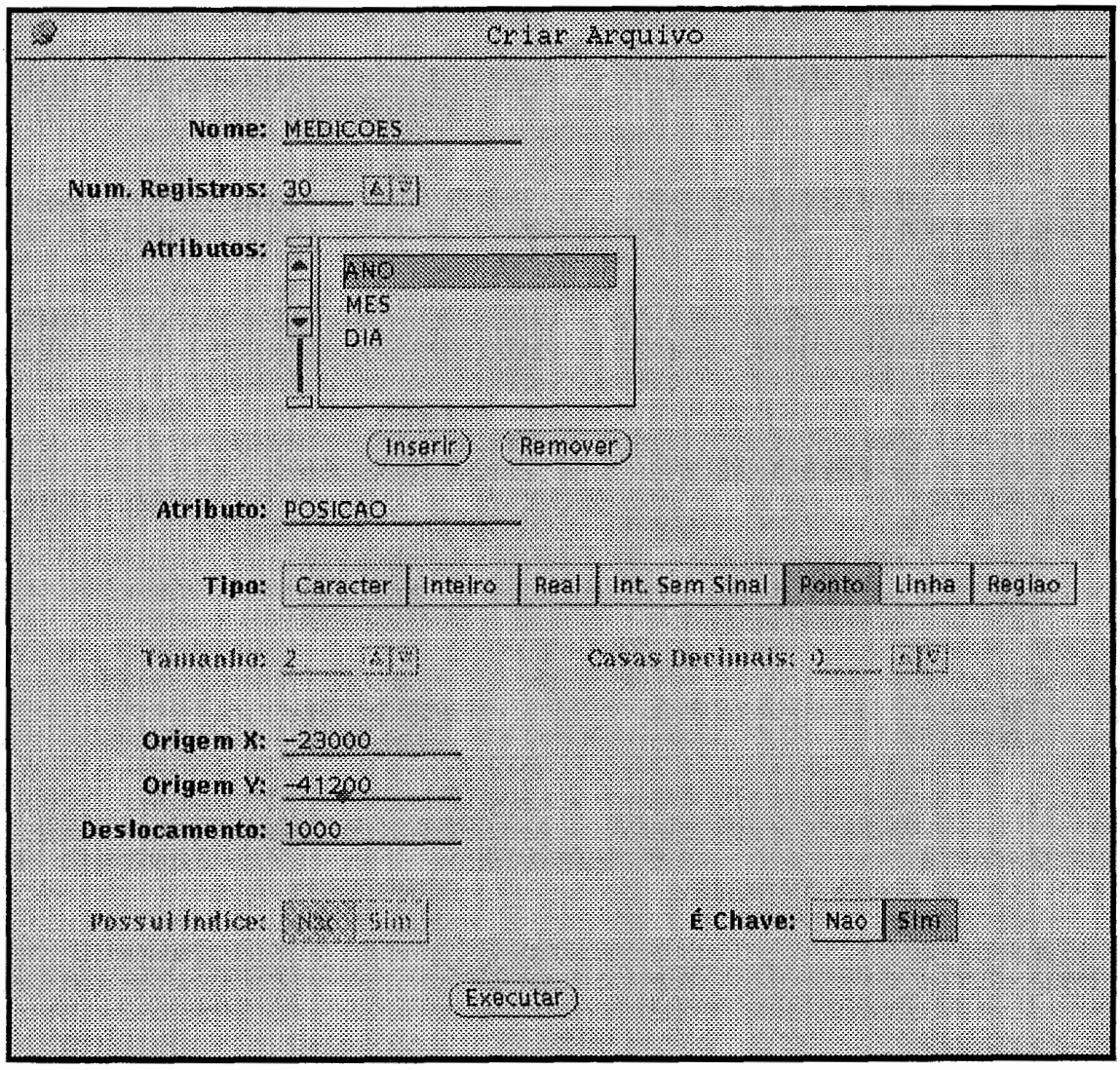


Figura 48 - Janela do comando criar arquivo do PCE

Algoritmo

Manda o COPPEREL criar *arquivo1*. Sendo que os atributos espaciais serão criados como inteiros sem sinal, sendo que o do tipo ponto será sempre indexado.

SE não houver erro
    PARA cada atributo espacial definido
        SE o atributo for do tipo *ponto*
            Cria uma *PR quadtree* que conterá os pontos do atributo
        FIMSE
        Insere o atributo na tabela ATR_ESPACIAIS (esquema espacial)
    FIM
    Insere a tabela *arquivo1* em REL_ESPACIAIS (esquema espacial)
SENÃO
    Envia Mensagem de ERRO
FIMSE

3.4.4.15. Comando relacional ORDENAR REGISTROS envolvento relação espacial

Este comando na linguagem LOPEREL* pode ser escrito da mesma forma que na linguagem LOPEREL.

ordenar registros de *arquivo1* por *atributo1*$_{1,1,1}$

Este comando é entrado pelo usuário através da janela apresentada na figura 49, onde são listados apenas os nomes dos arquivos que o usuário possui permissão para remover. Quando o usuário seleciona a relação a ser ordenada, o sistema mostra todos os seus atributos para que o usuário possa selecioná-los. Por exemplo, na figura 49 o usuário esta ordenando a tabela *medicao* pelos atributos *mes* e *dia*. O PCE não permite que o usuário selecione algum atributo espacial. Não foi implementado nenhum tipo de ordenação para os atributos espaciais. Por exemplo, poderíamos pensar em ordenar os pontos segundo a proximidade deles em relação a origem.

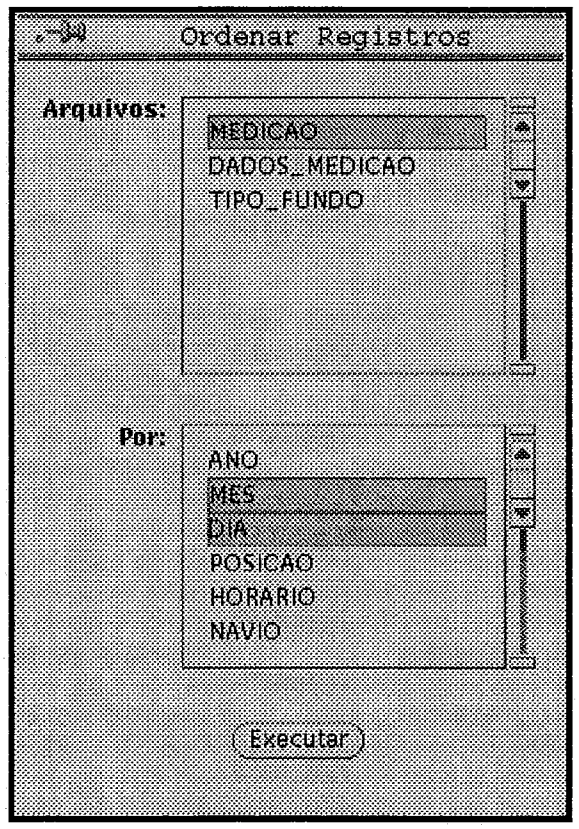


Figura 49 - Janela do comando ordenar registros do PCE

Para o COPPEREL os atributos do tipo ponto, linha e região são apenas atributos do tipo inteiro sem sinal, sendo portanto ordenáveis. No entanto, a ordenação desses atributos não traria nenhuma ordenação para o GEOCOPPE. Para os pontos estaríamos apenas ordenando os seus códigos, para as linhas e regiões estaríamos apenas ordenando o nome delas.

3.4.4.16. Comando relacional MOSTRAR envolvento relação espacial

Este comando na linguagem LOPEREL* pode ser escrito da mesma forma que na linguagem LOPEREL.

mostrar *arquivo1*

O comando mostrar é fornecido através da janela da figura 50, onde são listados todos os arquivos e variáveis do tipo ponto que o usuário possui permissão para ler. Só são permitidos que sejam mostrados os arquivos e variaveis não espaciais, as variáveis do tipo ponto, e os arquivos espaciais que possuem apenas atributos não espaciais e/ou atributos espaciais do tipo ponto. Quando o usuário pressiona o botão executar é aberta uma nova janela que contém todas as tuplas da relação selecionada. Por exemplo, a figura 50 o usuário esta querendo ver as tuplas da relação medicao, que são vistas através da janela aberta pelo sistema da figura 60.

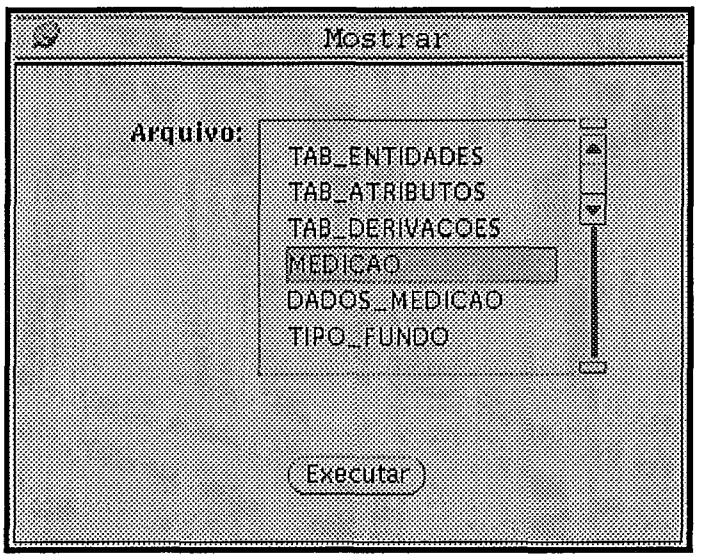


Figura 50 - Janela do comando mostrar do PCE

3.4.4.17. Comando relacional EXIBIR envolvento relação espacial

Este comando na linguagem LOPEREL* pode ser escrito do modo:

exibir *arquivo* o atributo *atributo*

O comando exibir não existe no COPPEREL, foi criado no GEOCOPPE para que pudessem ser mostrados ao usuário as relações espaciais que possuem algum atributo do tipo linha ou região e as variáveis do tipo linha ou região. Este comando é fornecido através da janela da figura 51, onde são listados todos os arquivos que o usuário possui permissão para ler. A janela deste comando é a única do GEOCOPPE que não possui o botão *executar.* O usuário seleciona uma

relação da lista de relações apresentada na janela, e um atributo da lista que contém os atributos da relação selecionada. Neste momento, o atributo é exibido. Se o atributo selecionado for do tipo linha ou região, este é exibido na área de desenho localizada na parte de baixo da janela. Os demais tipos são exibidos no campo *valor*. Por exemplo, na figura 51 o usuário esta querendo que seja exibido o atributo região da relação *contem*.

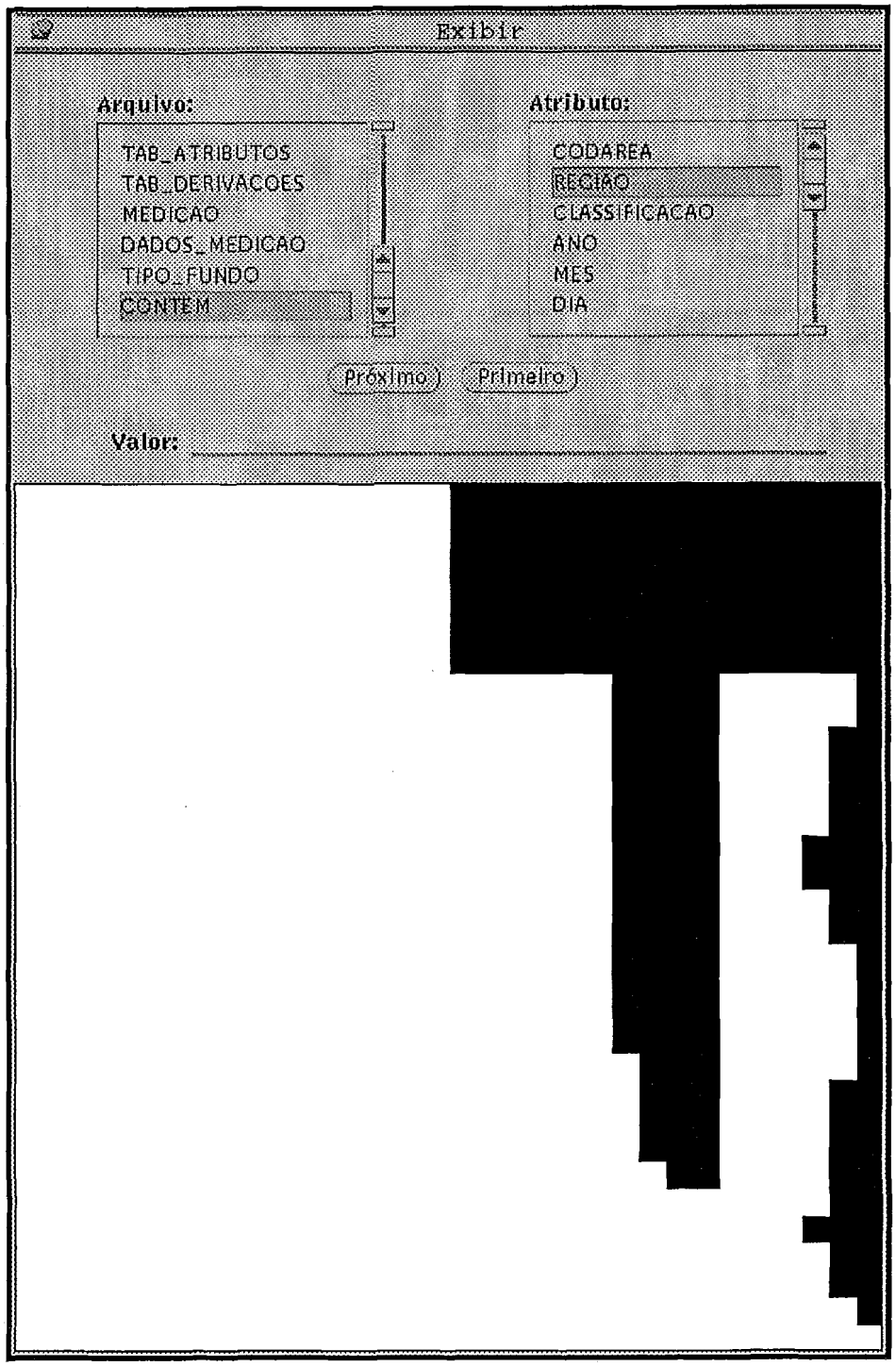


Figura 51 - Janela do comando exibir do PCE

3.4.4.18. Comando relacional INSERIR REGISTROS envolvento relação espacial

Este comando na linguagem LOPEREL* pode ser escrito da mesma forma que na linguagem LOPPEREL.

inserir registros *arquivo1*

O comando inserir registros é fornecido através da janela da figura 53, onde são listados todos os arquivos que o usuário possui permissão para escrever.

Para inserir um registro num arquivo através desta janela, o usuário primeiramente seleciona o arquivo no qual será feita a inserção. Neste momento, os atributos deste arquivo são listado ao lado. O usuário seleciona então um atributo, e, dependendo do seu tipo, alguns campos se apresentarão esmaecidos. Por exemplo, na figura 53 foi selecionado um atributo do tipo região, portanto os campos que não são utilizados para fazer a entrada dos valores deste tipos estão esmaecidos.

Os valores dos atributos relacionais são fornecidos através do campo *valor* localizado logo abaixo das listas. As coordenadas do tipo ponto são fornecidas através dos campos *X* e *Y*. As regiões são fornecidas com o auxílio de um arquivo, que conterá uma imagem representada por 0 e 1. A primeira linha deste arquivo contém o número de linhas e de colunas da imagem, e o nível máximo da *quadtree* que pode ser obtida com a imagem. Por exemplo, para a figura 53 apresenta uma região obtida através do arquivo solo1, mostrado abaixo.

```
32  32  5
00000000000000001111111111111111
00000000000000001111111111111111
00000000000000001111111111111111
00000000000000001111111111111111
00000000000000001111111111111111
00000000000000001111111111111111
00000000000000001111111111111111
00000000000000000000001111000001
00000000000000000000001111000001
00000000000000000000001111000011
00000000000000000000001111000011
00000000000000000000001111000011
00000000000000000000001111000111
00000000000000000000001111000111
00000000000000000000001111000011
00000000000000000000001111000011
00000000000000000000001111000001
00000000000000000000001111000001
00000000000000000000001111000001
00000000000000000000001111000001
00000000000000000000000111000001
00000000000000000000000111000011
00000000000000000000000111000011
00000000000000000000000111000011
00000000000000000000000011000011
00000000000000000000000000000011
00000000000000000000000000000111
00000000000000000000000000000011
00000000000000000000000000000011
00000000000000000000000000000001
00000000000000000000000000000000
```

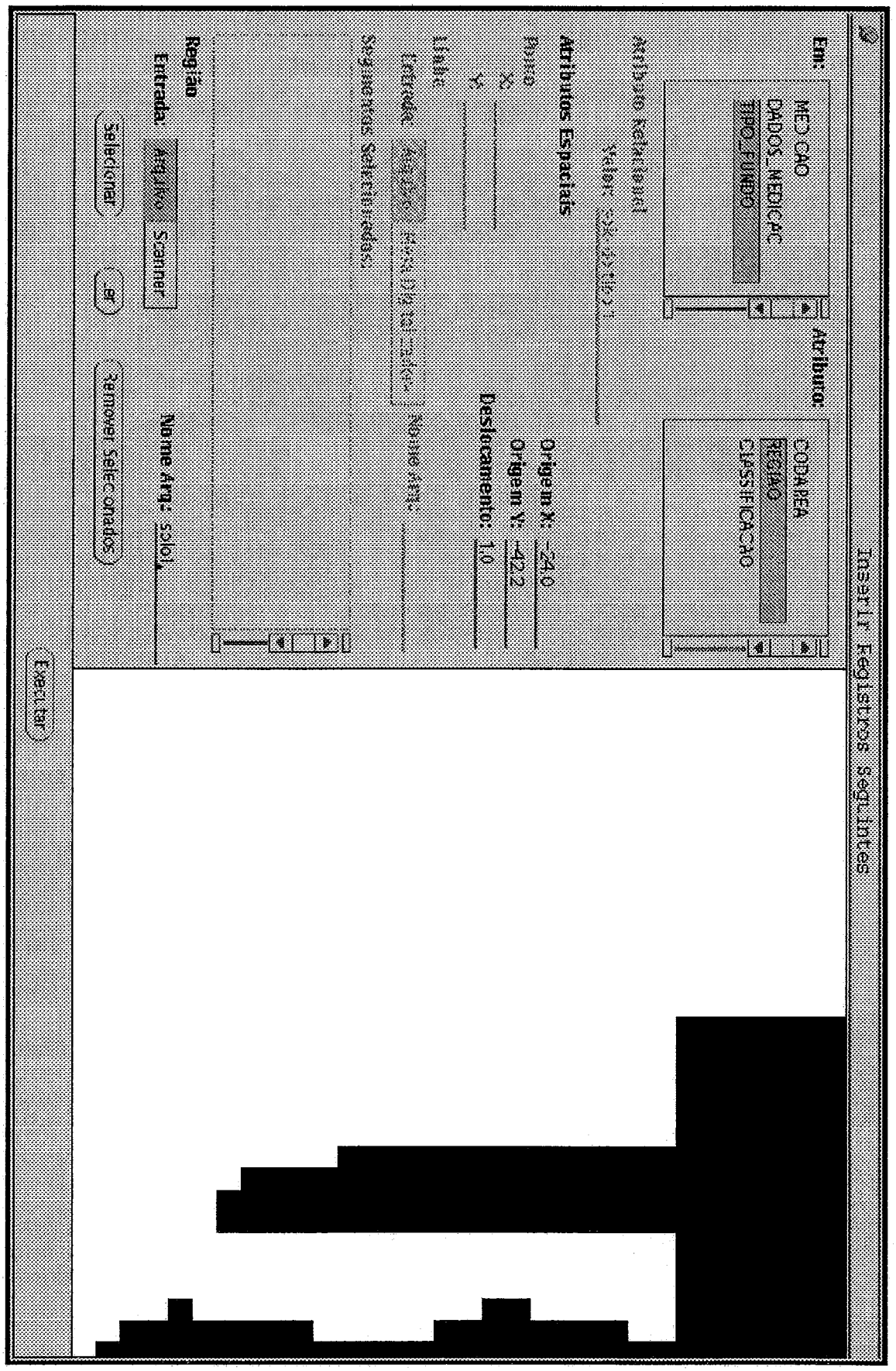


Figura 52 - Janela do comando inserir registros do PCE, onde esta sendo inserida a primeira tupla do arquivo *tipo_fundo*.

3.4.4.19. Comando relacional INSERIR VARIAVEL envolvento atributo espacial

Este comando na linguagem LOPEREL* pode ser escrito da mesma forma que na linguagem LOPPEREL.

inserir variavel *variavel1*

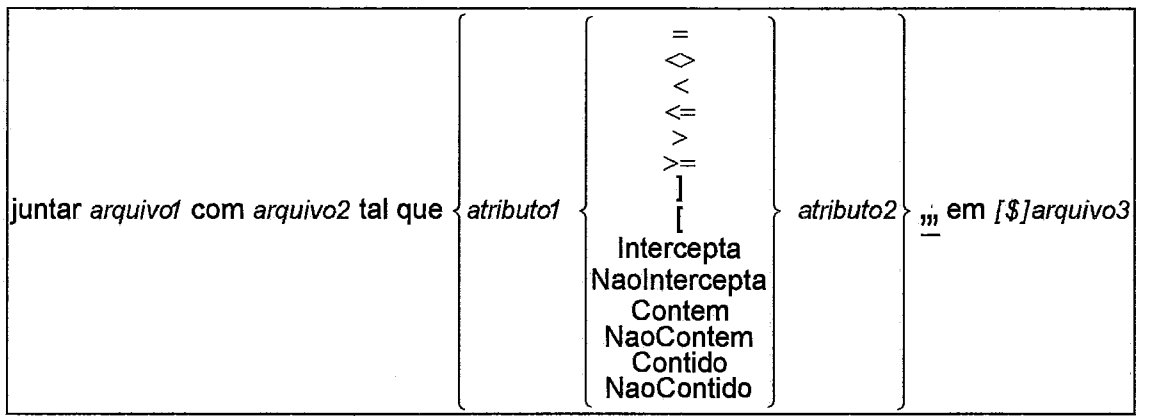
O comando inserir variavel é fornecido através da janela da figura 53, onde são listados todos as variaveis que o usuário possui permissão para escrever.

Para inserir um valor numa variável através desta janela, o usuário primeiramente seleciona a variável na qual será feita a inserção. Dependendo do seu tipo, alguns campos se apresentarão esmaecidos. Por exemplo, na figura 53 foi selecionado uma variável do tipo região (AREIA), portanto os campos que não são utilizados para fazer a entrada dos valores deste tipos estão esmaecidos. A entrada dos valor da variável é feita de modo similar ao realizado no comando inserir registros.

Figura 53 - Janela do comando inserir variavel do PCE.

3.4.4.20. Comando relacional JUNTAR envolvento relação espacial

Este comando na linguagem LOPEREL* pode ser escrito da mesma forma que na linguagem LOPEREL.



Na descrição acima do comando juntar, o que está especificado dentro das chaves mais externas (*atributo1* operador *atributo2*), será chamado de fator da junção. Portanto, uma condição de junção é formada por vários fatores de junção.

Este comando é fornecido através da tela apresentada na figura 54, onde o usuário realiza a junção da relação *tipo_fundo* com a relação *medicao* e o resultado é colocado na relação temporária *contem*. Neste caso, foi especificado na condição de junção o operador espacial *contem*. Para poder se ver o resultado desta junção foi feito o "copiar correspondendo", da figura 27, sobre a tabela *contem* obtendo-se a tabela *contem2*, apresentada na figura 56.

A figura 55 apresenta uma outra junção que também envolve relações espaciais. O resultado desta junção é colocado na relação temporária *temp1*.

Na junção, ocorre basicamente o mesmo problema encontrado no comando selecionar, ao tentar implementá-la sem alterar ou chamar diretamente as rotinas do COPPEREL. Para realizar a junção foram alteradas as rotinas VRFJUN e JUNTAR do COPPEREL. Além disto, foram desenvolvidas duas rotinas: a rotina CRCNCLE, que cria um arquivo concatenação compatível espacialmente com um outro arquivo passado como parâmetro de entrada para a rotina; a rotina CMPATRE que compara os dois atributos de um fator de junção, quando o operador especificado for espacial.

Algoritmo

SE *arquivo1* e *arquivo2* não forem espaciais
    Envia o comando para COPPEREL
SENÃO
    SE *arquivo3* já existir
        Verifica através da rotina VRUNCL se os arquivos *arquivo1, arquivo2* são concatenação compatíveis com o arquivo *arquivo3*
    SENÃO SE *arquivo3* for definido como arquivo temporário
        Chama a rotina CRCNCLE para criar *arquivo3* como sendo concatenação compatível espacialmente com os arquivos *arquivo1* e *arquivo2.*
    SENÃO
        ERRO - A relação resultado não existe
    FIMSE
    Aloca na base de dados, através da rotina ALCDSP do COPPEREL, uma área que conterá a lista dos registros selecionados de *arquivo1* e *arquivo2* na rotina JUNTAR
    Chama a rotina JUNTAR
    Chama a rotina LIBDSP para liberar a área da base de dados do COPPEREL alocada anteriormente
FIMSE

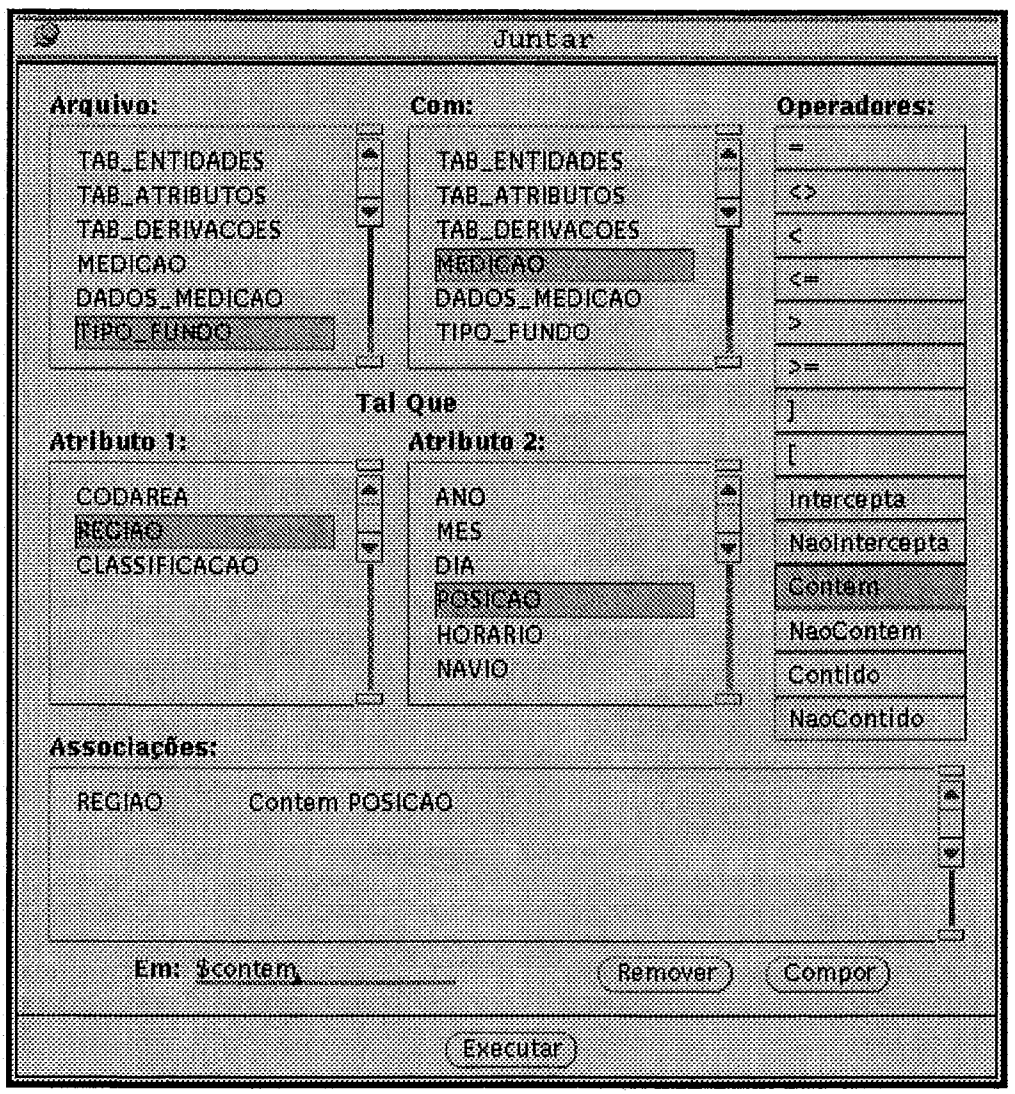


Figura 54 - Janela do comando juntar do PCE. Junção utilizando o operador espacial contem.

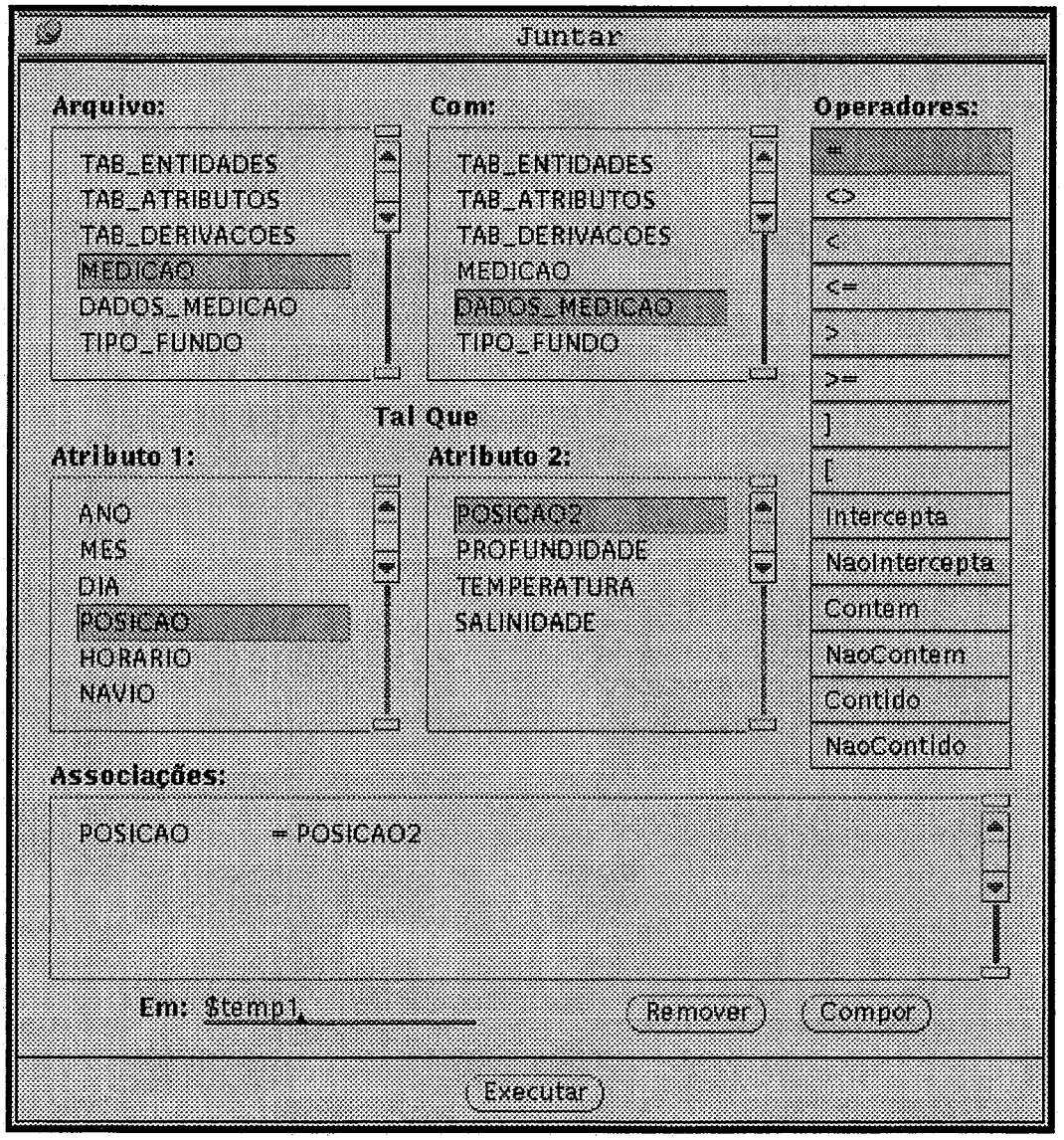


Figura 55 - Janela do comando juntar do PCE. Junção utilizando o operador espacial de igualdade.

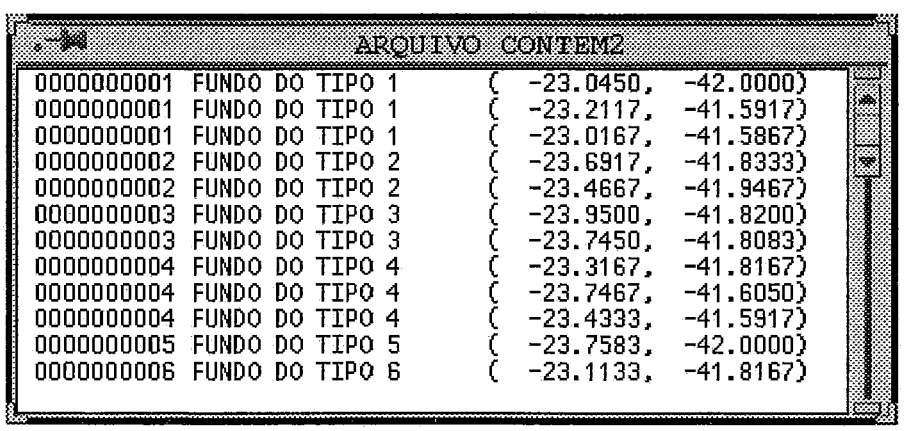


Figura 56 - Projeção da tabela *contem*, obtida na junção da figura 54

A operação de junção no COPPEREL, implementada pela rotina JUNTAR, é realizada em duas etapas. Na primeira os registros dos dois arquivos que satisfazem a condição de junção são inserido num lista. Na segunda os registros selecionados são juntados e o registro resultante inserido no arquivo resultado.

A rotina VRFJUN, chamada pela rotina JUNTAR, insere numa lista os registros que satisfazem a condição de junção. Esta rotina foi alterada para suportar os fatores de junção espaciais. Para resolver-los, a VRFJUN chama a rotina CMPATRE.

Na primeira etapa, originalmente a rotina JUNTAR realiza três tipos de processamento de junção. No primeiro as duas relações (*arquivo1* e *arquivo2*) são ordenadas antes de serem processadas. A ordenação só é utilizada quando numa das expressões da junção existir um dos seguintes operadores relacionais: =,<,>,<=,>=. No segundo processamento é utilizado um índice sobre uma das relações, ou a da esquerda (*arquivo1*) ou a da direita (*arquivo2*). O processamento por índice tem preferência sobre a ordenação quando a condição de junção utilizar o operador de igualdade e for definida sobre um atributo indexado. O terceiro processamento é a "força bruta", e só é aplicada quando nenhum dos outros dois tipos de processamento pode ser usado, ou se tiver ocorrido algum erro durante a ordenação das relações.

Para que a rotina JUNTAR processasse os dados espaciais, foi alterado somente o processamento por índices, já que a ordenação não pode ser aplicada aos dados espaciais (não faz sentido). O processamento por índice dos dados espaciais nunca tem preferência sobre o processamento por índice ou por ordenação dos dados relacionais. O processamento por indice dos dados espaciais só existe quando for definida uma expressão de junção do tipo *atrponto1* = *atrponto2*, ou do tipo *atrponto1* contido *região2*, ou do tipo *região1* contem *atrponto2*, onde *atrponto1* e *atrponto2* são atributos do tipo ponto, respectivamente, da relação *arquivo1* e *arquivo2*; e *região1* e *região2* são atributos do tipo região, respectivamente, da relação *arquivo1* e *arquivo2*. Caso contrário, é utilizado o processamento por "força bruta".

No primeiro caso, chama-se a rotina CODIGO_DOS_PONTOS_IGUAIS do GDE que faz a interseção das *PR quadtrees* que representam os dois atributos e devolve, numa área da base de dados do COPPEREL cujo endereço foi passado como parâmetro de entrada, uma lista com os pares dos códigos dos pontos dos dois atributos que são iguais (isto é, possuem a mesma coordenada, não o mesmo código de ponto).

No segundo e no terceiro caso, a rotina CODIGO_DOS_PONTOS_CONTIDOS_NA_REGIAO, do GDE, será chamada. Esta rotina realiza a interseção da *PR quadtree*, que indexa o atributo do tipo ponto, com cada uma das *quadtrees* de região existentes nas tuplas que representam o atributo do tipo região. Este processamento não tem preferência sobre o primeiro tipo de processamento (primeiro caso) que, por sua vez, não tem preferência sobre o processamento dos dados relacionais.

Na segunda etapa, quando os registros selecionados são "juntados", a rotina necessitou ser alterada de modo a copiar corretamente os atributos espaciais para a relação resultados. Isto é feito do seguinte modo:

```
SE existe algum atributo espacial no arquivo resultado
    PARA I=1 ATÉ número de atributos espaciais
        SE o tipo do atributo I for ponto
            Cria uma PR quadtree igual a utilizada para indexar este atributo I
        FIMSE
    FIMPARA
FIMSE
PARA I=1 ATÉ número de registros selecionados
    Busca os registros selecionados dos dois arquivos, colocando-os numa tupla (em
        memória)
    SE existir algum atributo espacial em qualquer um dos dois arquivos
        PARA J=1 ATE número de atributos espaciais (dos dois arquivos)
            SE o atributo J for do tipo ponto
                Recupera a coordenada do ponto (cujo código é o valor do atributo J)
                Insere o ponto na PR quadtree que indexa o atributo na relação resul-
                    tado, obtendo o código do ponto (cod)
                Copia o código cod para a tupla da relação resultado
            SENÃO
                Copia a quadtree (quad1) que o representa para uma nova quadtree
                    (quad2)
                Copia o nome de quad2 para a tupla da relação resultado
            FIMSE
        FIMPARA
        Insere a tupla montada na relação resultado.
    FIMSE
FIMPARA
```

Se a relação resultado já existir, esta não será desalocada antes da junção. As tuplas obtidas são inseridas numa área da base de dados, sendo que o endereço antigo da relação só é substituido pelo endereço desta área, quando a junção for concluída com sucesso.

3.4.4.21. Comando relacional SELECIONAR envolvento relação espacial

Este comando na linguagem LOPEREL* pode ser escrito do modo:

selecionar [registros de] *arquivo1* tal que *condição* em [$]*arquivo2*

onde:

condição := *fator-condicional* [ou *fator-condicional*] ...

fator-condicional := [[area | perimetro]] *nome-do-atributo* { falso | = | <> | < | <= | > | >= | [|] | Intercepta | NaoIntercepta | Contem | NaoContem | Contido | NaoContido | verdadeiro } { [-] { *cadeia* | *inteiro* | *numero* } | [[area | perimetro]] { *nome-da-variavel* | *(x1,y1;x2,y2)* | *nome-do-atributo* } | [perimetro] *(x,y)* *(x1,y1);(x2,y2)* } eee

Obviamente, nem todas as combinações são possíveis. Dependendo do tipo da variavel(is) especificada(s) a combinação será ou não permitida. Por exemplo, não é possível aplicar a função perimetro sobre um atributo do tipo ponto ou um atributo não espacial. O comando selecionar é fornecido através da janela mostrada na figura 57, onde são listados apenas os arquivos que o usuário possui permissão par ler. O usuário fornece os *fatores-condicionais* da condição de seleção através dos botão compõe. Se o usuário tentar fornececer uma composição para o fator condicional que não seja válida, o sistema devolve uma mensagem de erro. Os boleandos e e ou que unem os *fatores-condicionais* são fornecidos através do botão Op. Lógicos. O botão boleanos é utilizado para fornecer os dois operadores boleandos verdadeiro e falso. O botão remove a condição de seleção que está sendo construída.

Na figura 57 esta sendo fornecida uma seleção sobre a relação *temp1*, que foi obtida através da junção da figura 55. O resultado desta seleção, o arquivo *temp2* é mostrado na figura 58.

A princípio este comando foi projetado de modo a não ser necessário alterar a rotina SLCNAR do COPPEREL, responsável pela seleção dos registros, não sendo feitas chamadas diretas a rotinas do COPPEREL. Para tanto, era indispensável que o usuário fornecesse duas condições de seleção, e um operador lógico para uní-las. A primeira condição de seleção conteria as expressões que envolvessem relações não espaciais, e a segunda condição conteria as expressões que envolvessem relações, operadores e funções espaciais. Seriam então realizadas duas operações de seleção, uma para cada condição de seleção, e o resultado depositado em duas relações temporárias distintas. O PCE chamaria o COPPEREL para resolver a primeira seleção, que conteria a condição relacional. A segunda seria resolvida pelo próprio PCE, através de chamadas ao COPPEREL e ao GDE. Finalmente, o PCE uniria as duas relações resultado obtidas através do operador lógico especificado.

A solução apresentada acima possui dois inconvenientes. O primeiro incoveniente é que o usuário deve fornecer duas condições de seleção ao invés de apenas uma. O segundo incoveniente é que são executadas duas seleções diferentes sendo necessário um processamento posterior para unir as duas relações obtidas.

Para evitar que isto fosse necessário, a rotina SLCNAR foi alterada de forma a realizar uma seleção na qual a condição de seleção envolva relações espaciais. Para tanto, foi necessário aumentar o número de informações passadas como parâmetro para a rotina SLCNAR, através da via expressa do COPPEREL . A via expressa é um vetor (*viaexp*) definido na estrutura *maquina*, e utilizado pelo COPPEREL para a passagem de parâmetros entre suas rotinas. Deste modo, a via expressa na rotina SLCNAR passou a ser descrita do seguinte modo:

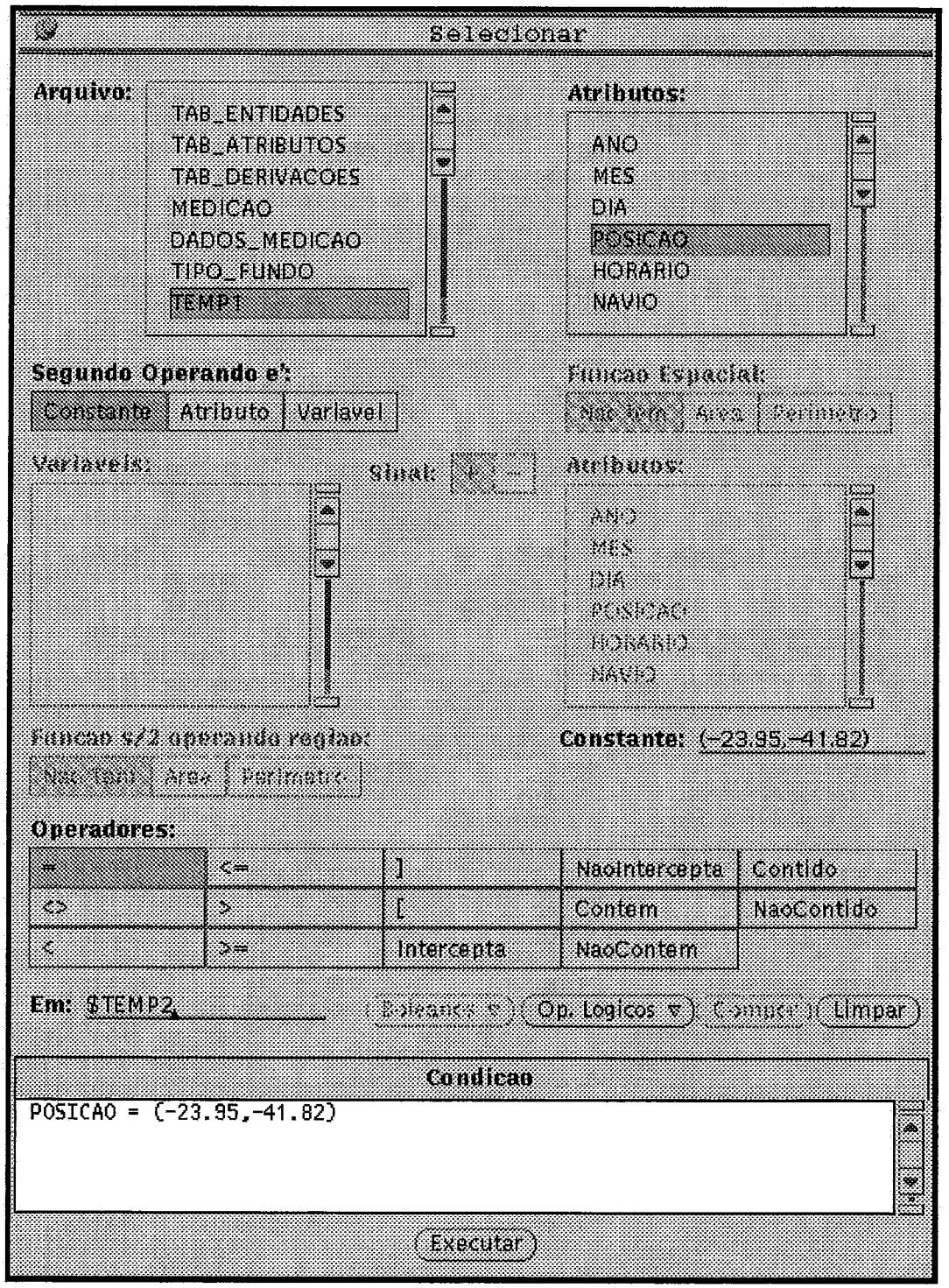
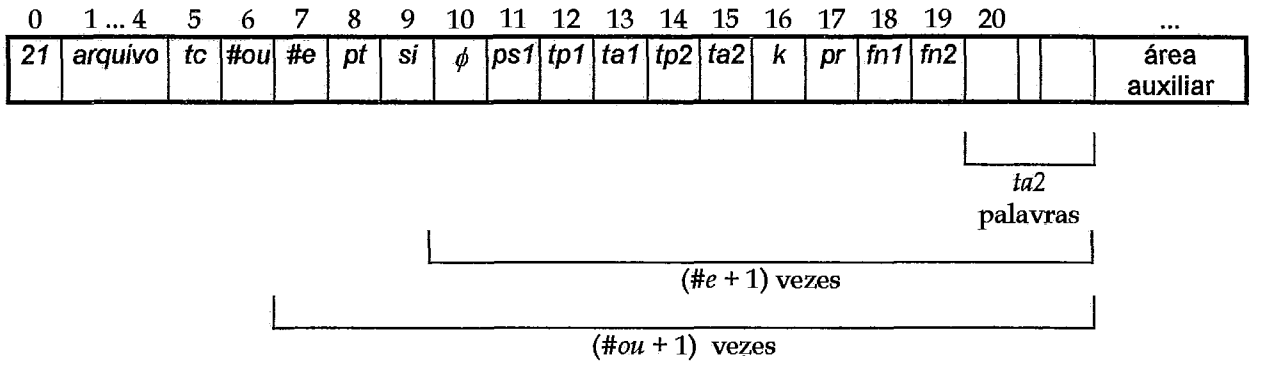


Figura 57 - Janela do comando selecionar do PCE

Figura 58 - Arquivo *temp2*, resultado de uma seleção sobre o arquivo *temp1*.



tc = tamanho da condição
= 0 => *maqina_1.viaexp[6]* = V ou F
> 0 => execução por índices
< 0 => execução por força bruta
tc + 7 => início de área auxiliar

#ou = número de "ou"s da condição (#ou + 1 ° números de fatores a seguir)

#e = número de "e"s do fator (#e + 1 ° número de comparações a seguir)

pt = aponta para a comparação sobre a qual será usado o índice

si = usado na execução do comando (para o endereço do índice)

ϕ = operador relacional *ou espacial.*

ps1 = posição do atributo no arquivo

tp1 = tipo do atributo do arquivo

ta1 = tamanho do atributo do arquivo.

tp2 = tipo da expressão a comparar com o atributo

ta2 = tamanho da expressão a comparar com o atributo

k = código para a expressão a comparar com o atributo

= 0=> constante
as *ta2* palavras seguintes contém o valor a comparar
SE *tp2* for espacial
SE *tp2 = ponto*
As duas palavras seguintes contém o ponto a comparar
17 -> coordenada X
18 -> coordenada Y
SENÃO
As quatro palavras seguintes contém a linha a comparar. Os pontos finais do segmento não precisam estar ordenados.
17 -> coordenada X do ponto P1
18 -> coordenada Y do ponto P1
19 -> coordenada X do ponto P2
20 -> coordenada Y do ponto P2
FIMSE
FIMSE

= ±1 => outro atributo do mesmo arquivo
a palavra seguinte contém posição deste atributo.
valor a comparar = sinal de *k* * valor do atributo

= ±2 => variável
a palavra seguinte aponta para a área onde se encontra o nome da variável
valor a comparar = sinal de k * valor da variável

>3

<- 3 => atributo de arquivo em percurso
a palavra seguinte contém a posição deste atributo, a posição do cursor em relação ao topo da pilha de cursores é | k | - 3.
valor a comparar = sinal de k * valor do atributo

prquad = Se tp1=PONTO contém a parte inteira do nome da pr-quadtree que indexa os pontos do atributo selecionado
fn1 = Indica se é aplicada alguma função espacial sobre o primeiro atributo
fn2 = Indica se é aplicada alguma função espacial sobre a expressão a comparar

Na condição de seleção o operador lógico E tem precedência sobre o operador lógico OU [Zakim85] . Isto justifica, a estrutura da via expressa que foi apresentada acima, onde para cada OU tem-se um conjunto de E.

Algoritmo de Seleção do PCE

```
SE arquivo1 não for espacial
    Envia o comando para o COPPEREL
SENÃO
    Aloca na base de dados, através da rotina ALCDSP do COPPEREL, uma área que
        conterá a lista dos registros selecionados.
    SE arquivo2 for temporária e não existir
        Cria uma relação temporária com nome arquivo2 igual a arquivo1
    SENÃO
        Verifica se os arquivos arquivo1 e arquivo2 são união compatíveis
            espacialmente, através da rotinas VRUNCL
    FIMSE
    Chama a rotina SLCNAR para selecionar os registros
    Chama a rotina COPIAR para copiar a relação resultado os registros selecionados.
    Chama a rotina LIBDSP para liberar a área da base de dados do COPPEREL alocada
        anteriormente
    SE houve erro de execução
        SE foi criada a relação temporária arquivo2
            Remove-a da base de dados
        SENÃO
            Desaloca-a da base de dados
        FIMSE
    FIMSE
FIMSE
```

O COPPEREL realiza a seleção de dois modos: o primeiro é através da utilização de algum índice existente sobre a relação *arquivo1*, e o segundo é a "força bruta" (seleção tupla a tupla). Obviamente, o primeiro método é o mais indicado, mas só pode ser aplicado quando **em todos** os blocos de OU da condição de seleção existir alguma expressão sobre algum atributo indexado, e o operador relacional utilizado nesta expressão for a igualdade. Caso contrário será aplicada, por razões óbvias, a "força bruta".

Na seleção por índices o GEOCOPPE obtém proveito das *PR quadtrees*, usando-as quando possível como um índice sobre os atributos do tipo ponto. Isto pode acontecer quando o usuário especificar na condição de seleção uma fator condicional do tipo *atrponto = (x,y)* ou do tipo *atrponto* contido *(x1,y1;x2,y2)*, onde *atrponto* representa um atributo do tipo ponto.

No primeiro caso, a rotina SLCNAR vê se a *PR quadtree* que indexa o atributo *atrponto* possui o ponto *(x,y)*, isto é realizado procurando na relação que armazena os pontos da *PR quadtree* o código do ponto *(x,y)*. Se este não for encontrado significa que o ponto não existe. Caso contrário, o código é utilizado para encontrar as tuplas que satisfazem o fator condicional. É importante recordar que todo atributo do tipo ponto é criado no COPPEREL como um inteiro-sem-sinal indexado.

No segundo caso, a rotina SLCNAR manda o GDE realizar a interseção da *PR quadtree* que representa os pontos do atributo *atrponto* com a região *(x1,y1;x2,y2)* especificada. Isto é feito chamando a rotina CODIGO_DOS_PONTOS_CONTIDOS_NA_REGIAO do GDE, que coloca numa área da base de dados do COPPEREL, cujo endereço é passado como parâmetro de entrada para a rotina, uma lista com todos os códigos dos pontos da *PR quadtree* que se encontram dentro da região especificada. Do mesmo modo que no caso anterior, estes códigos são utilizados para encontrar as tuplas que satisfazem a expressão.

3.4.4.22. Comando relacional FECHAR BASE DE DADOS

Este comando na linguagem LOPEREL* pode ser escrito do mesma forma que na linguagem LOPEREL.

fechar base de dados { atualizando | nao atualizando }

Não existe uma tela especifica para este comando, pois não existem dados a serem fornecidos pelo usuário. Ele é simplesmente ativado através do menu do botão de *sessão.*

Quando o usuário escolhe a opção de fechar a base de dados atualizando, o PCE antes de fechá-la remove-se todas as *quadtrees* que estão associadas as relações temporárias. Isto é feito pela rotina FECHAR_BASE_ESPACIAL.

4. Exemplo utilizado

Para exemplificar esta tese foram criadas três relações, possuindo dois dos três tipos espaciais desenvolvidos (ponto e região). Estas relações são descritas através da figura 59, que contém os comandos criar arquivo da LOPEREL* utilizados para criar as relações.

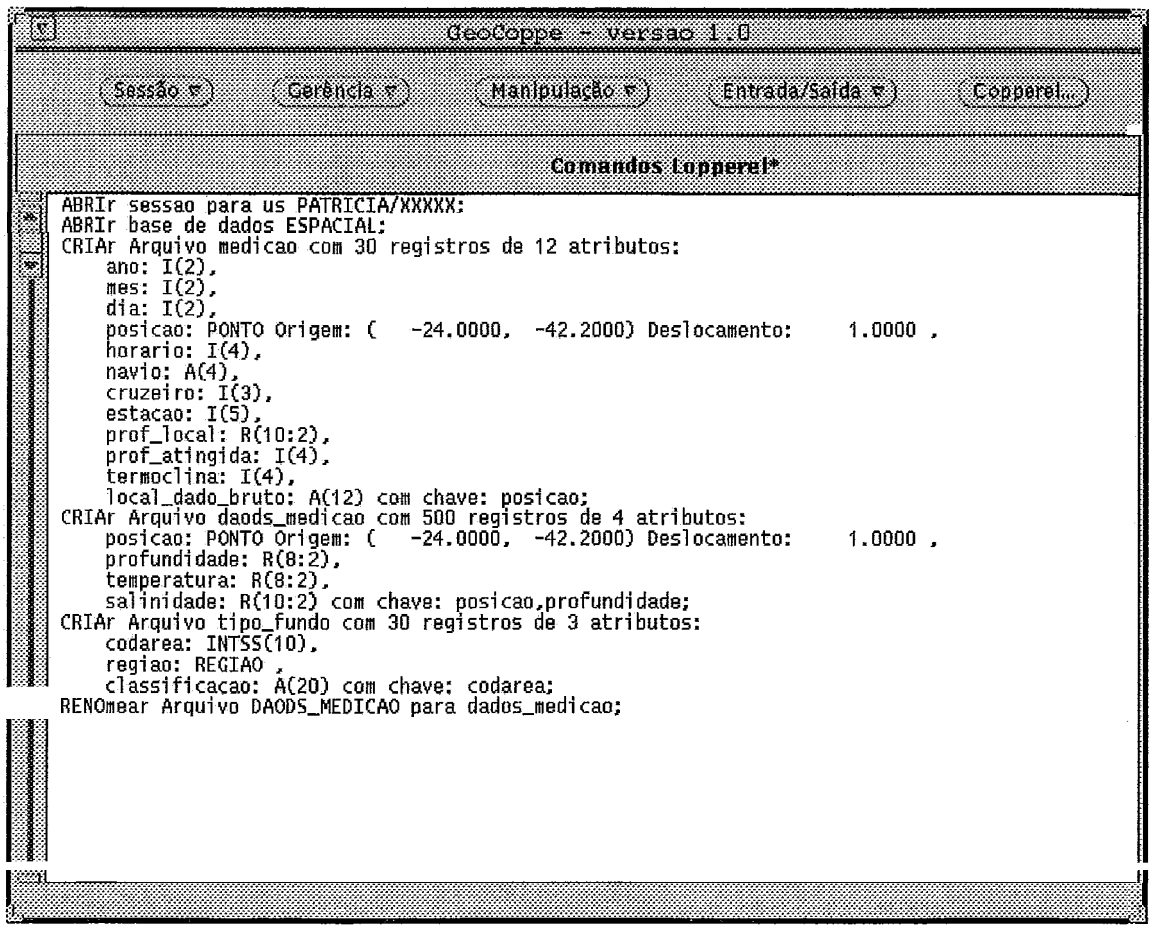


Figura 59 - Descrição dos arquivos utilizados com exemplo.

A base de dados empregada refere-se a dados oceanográficos como por exemplo, localizaçao através das coordenadas geográficas e profundidade; propriedades físico-químicas tais como: temperatura, salinidade, tipo de material do fundo, etc.

Os dados existentes nas tabelas *medicao* e *dados_medicao* foram cedidos pela Diretoria de Hidrografia e Navegação, DHN, e correspondem a medições realizadas por navios durante o ano de 1991. Tais dados foram coletados nos quadrados de Mardsen 30 e 31, que se situam aproximadamente na zona de exploração petrolífera no litoral do estado do Rio de Janeiro. Nas tabelas a seguir o atributo *posicao* representa a latitude/longitude do ponto de coleta do dado. Os dados da tabela *tipo_fundo* são fictícios. A tabela *medicao* é mostrada na

figura 60, a tabela *dados_medicao* e mostrada nas figuras 61-62, e as regiões da tabela *tipo_fundo* são mostradas nas figuras 53 e 63-67.

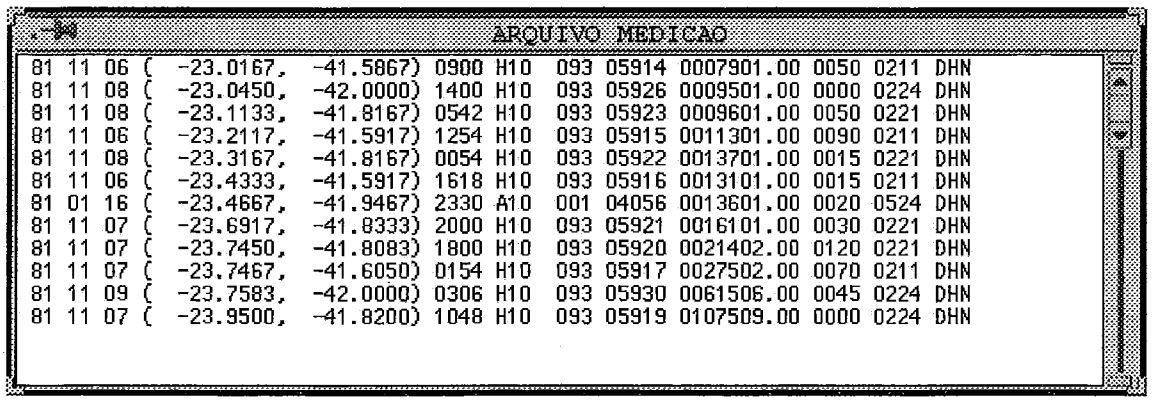


Figura 60 - Arquivo *medicao*.

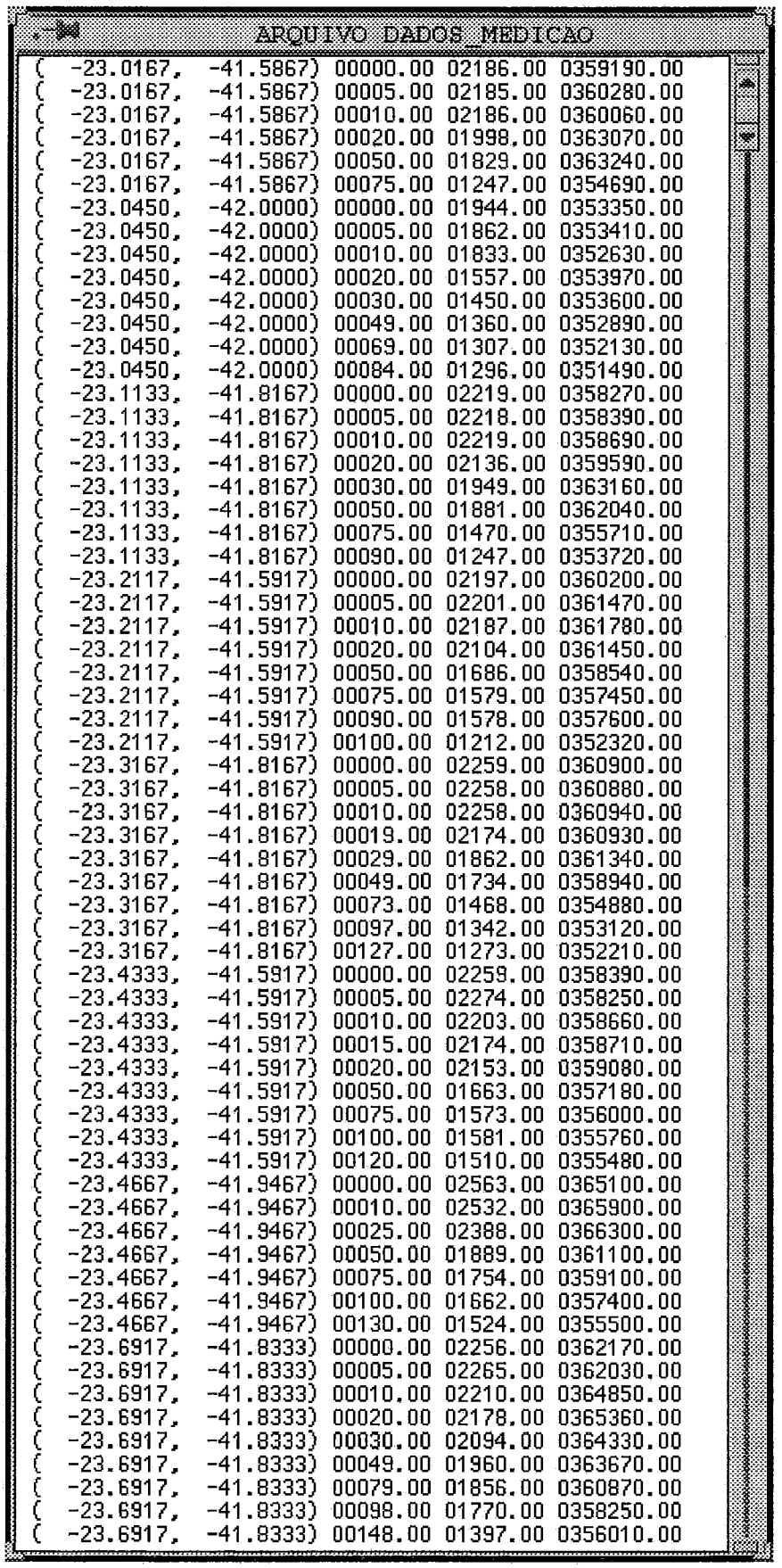


Figura 61 - Primeira parte do arquivo *dados_medicao*.

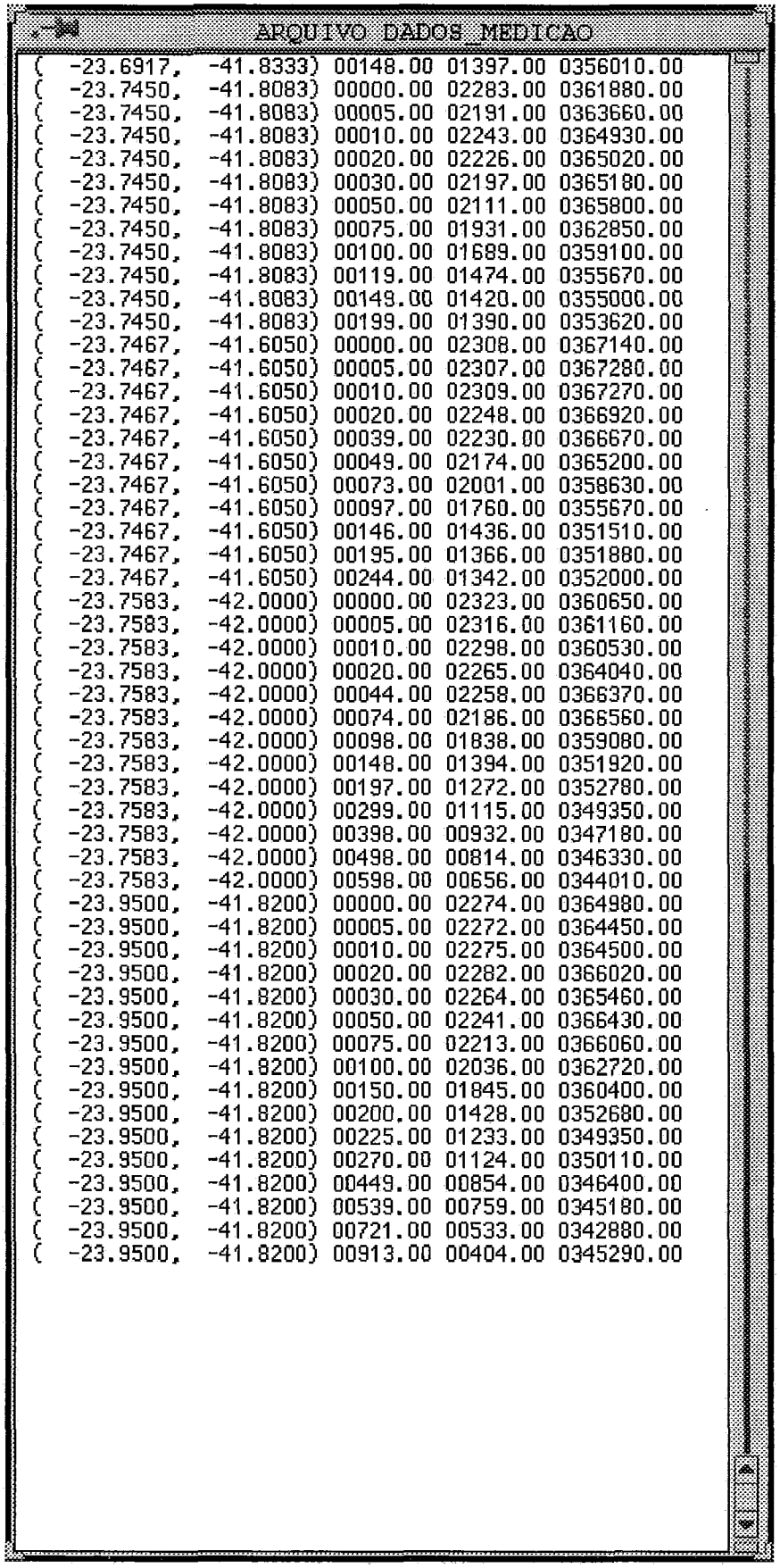


Figura 62 - Segunda parte do arquivo *dados_medicao*.

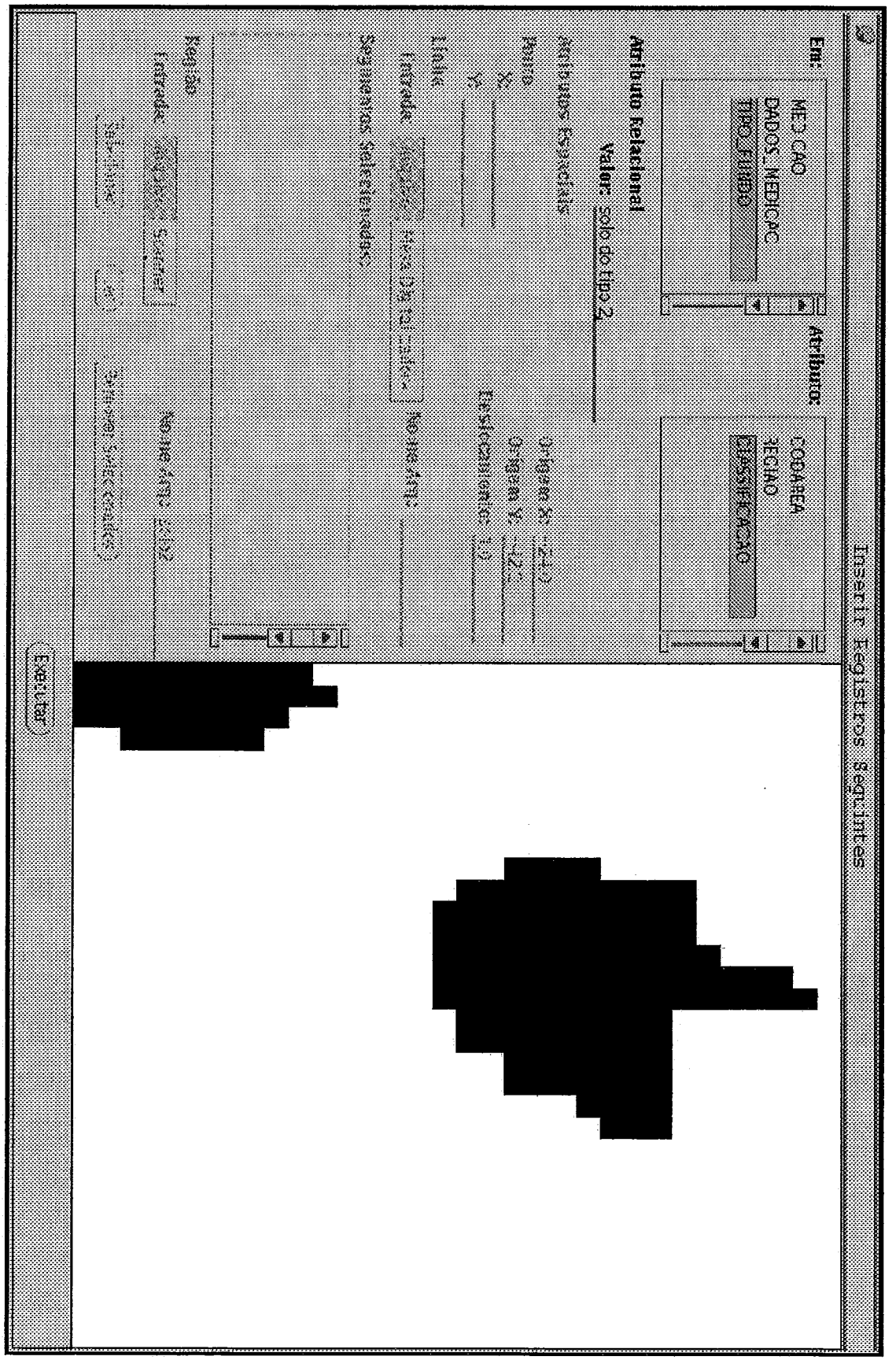


Figura 63 - Segunda tupla do arquivo *tipo_fundo*.

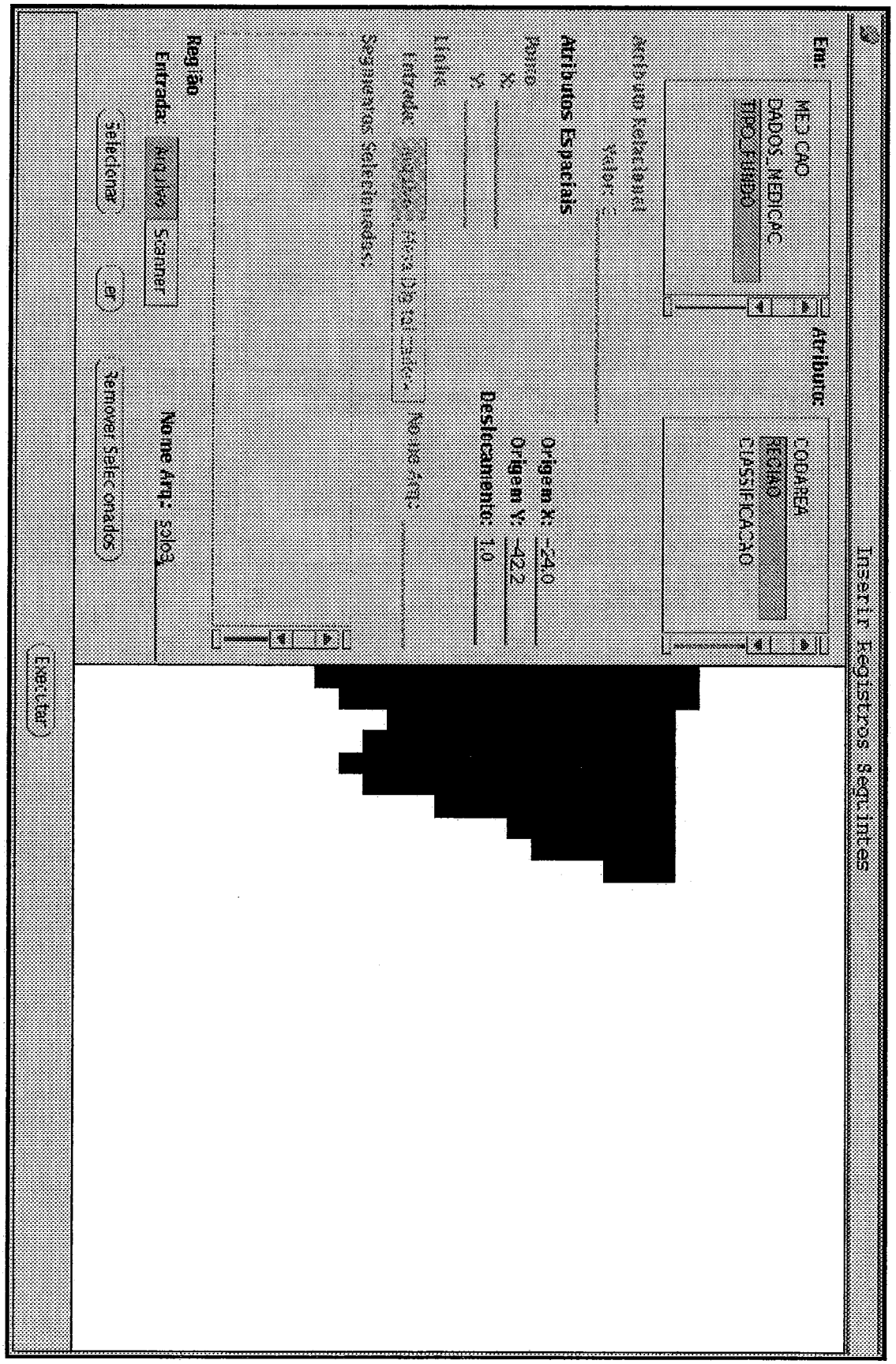


Figura 64 - Terceira tupla do arquivo *tipo_fundo*.

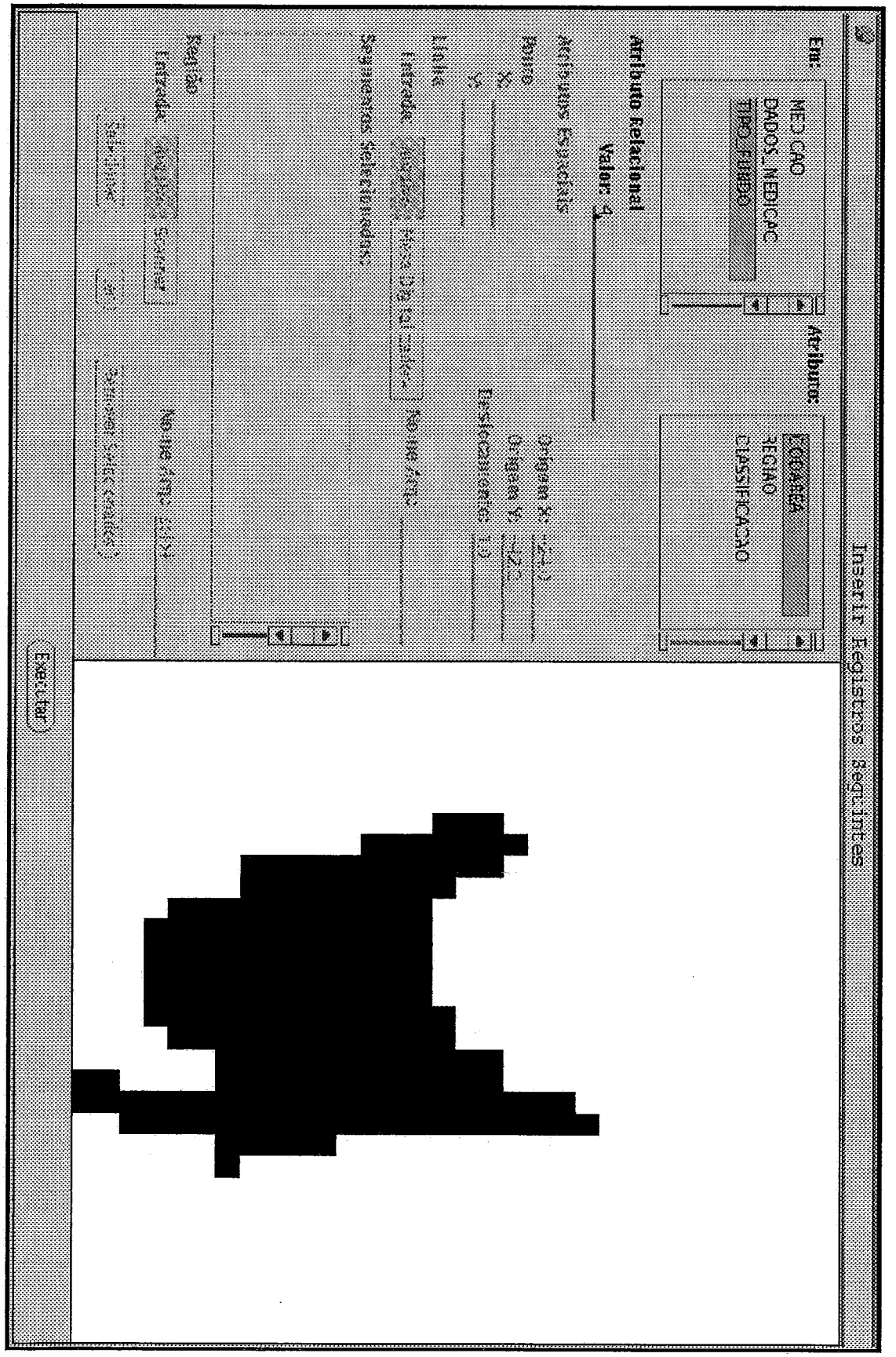


Figura 65 - Quarta tupla do arquivo *tipo_fundo*.

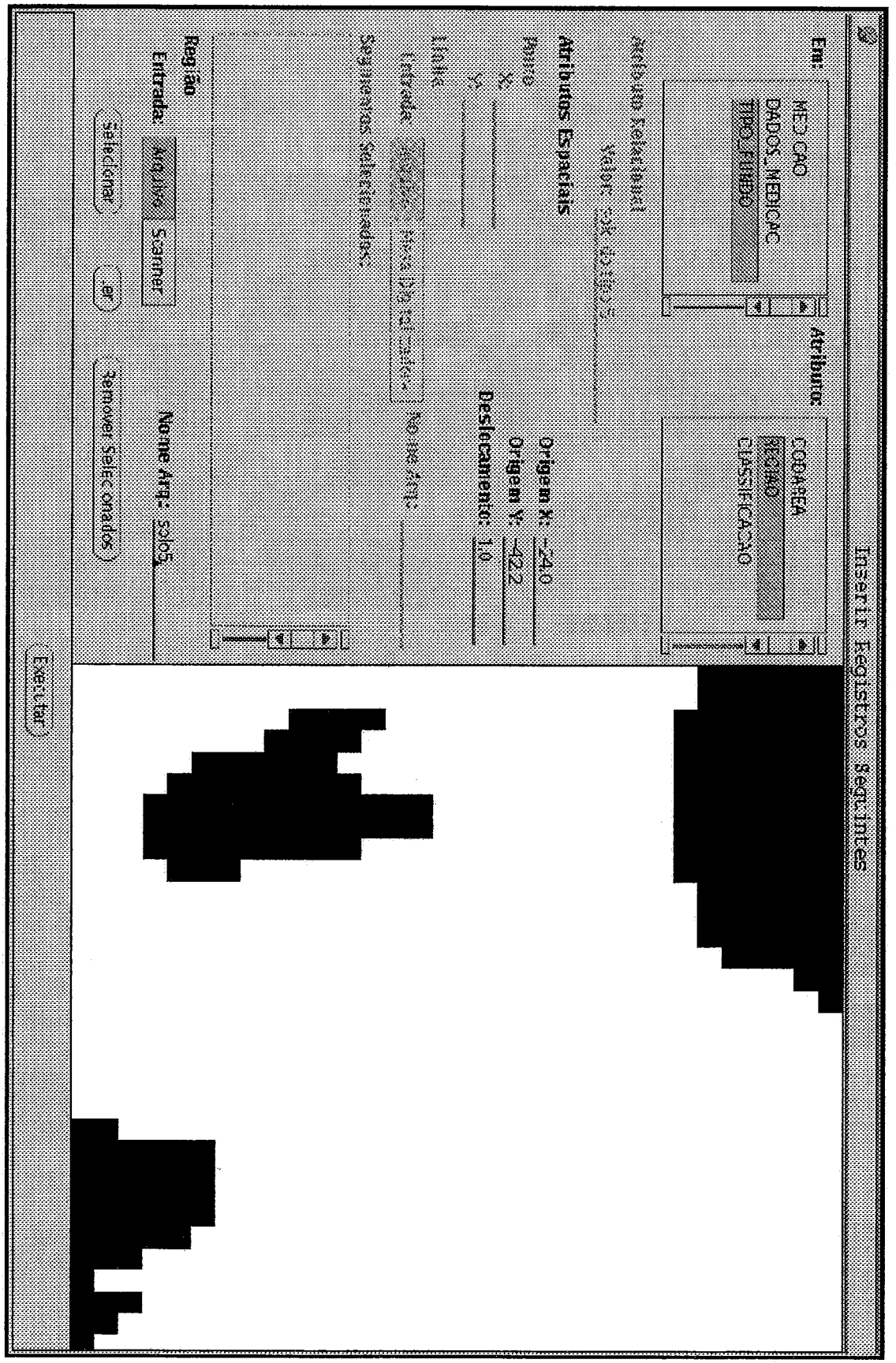


Figura 66 - Quinta tupla do arquivo *tipo_fundo*.

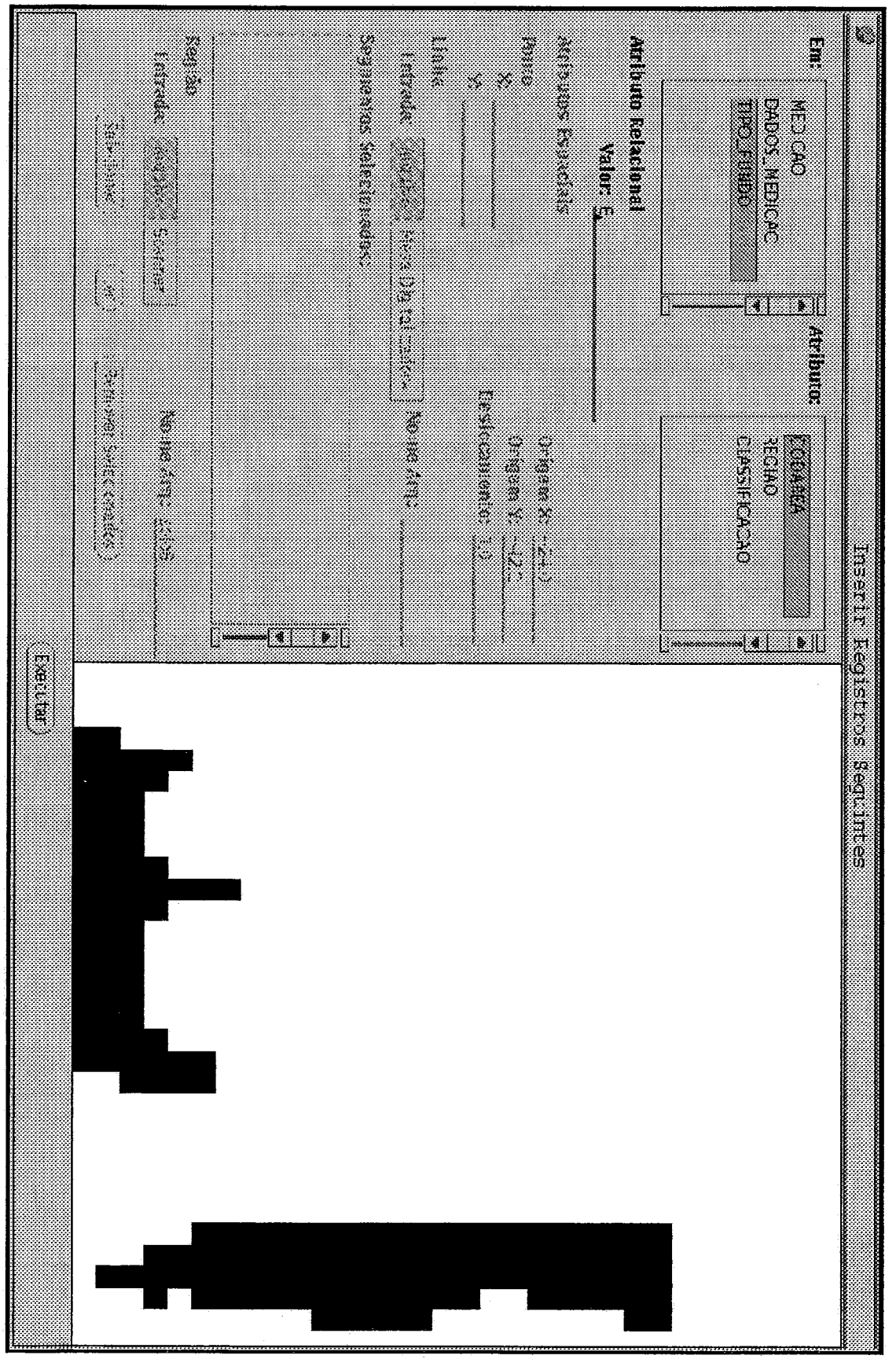


Figura 67 - Sexta tupla do arquivo *tipo_fundo*.

5. Conclusões e Recomendações

Partindo do banco de dados relacional, COPPEREL, desenvolvido na década de 80 no Programa de Engenharia de Sistemas, criou-se um Gerenciador de Dados Espaciais, GDE, e um Processador de Consultas e Exibição, PCE, objetivando desenvolver um sistema de banco de dados espacial, GEOCOPPE.

O GEOCOPPE incorporou o sistema COPPEREL como um de seus componentes, ampliando consideravelmente as suas potencialidades. O sistema GEOCOPPE é uma proposta de SGBD espacial que reune os três componentes COPPEREL, GDE e PCE de tal modo que o COPPEREL não necessitou ser alterado na sua filosofia relacional para suportar os dados espaciais. Foram acrescentados ao COPPEREL quatro novos tipos de dados: ponto, linha, região e inteiro sem sinal; sendo mantidas todas as suas funcionalidades e controles operacionais inalterados. Tal só foi possível em decorrência do uso de *quadtrees*. Pelo que se sabe o GEOCOPPE é o primeiro SGBD desenvolvido no Brasil que utiliza as estrutura de dados *quadtree*.

O fato das *quadtrees* terem sido armazenadas dentro de relações do COPPEREL, evitou a necessidade de manter qualquer outro tipo de estrutura ou dados fora do mesmo, como é comum acontecer quando se tem SGBDs espaciais montados sobre SGBDs relacionais.

O uso de *quadtrees* mostrou-se altamente eficiente para uso em SGBDs, pois estas estruturas levam a uma economia de espaço considerável, por permitirem agrupar, sob a mesma unidade de representação, folha, uma grande quantidade de informações espaciais que possuam propriedades idênticas (bloco grande). Usando as *quadtrees* ao invés da usual estrutura vetorial o SIG SPANS, pioneiro no uso de *quadtrees* em sistemas comerciais, consegue uma redução de 3 a 8 vezes no espaço de armazenamento.

No GEOCOPPE, as *quadtrees* são armazenadas como *quadtrees* lineares, o que facilita o gerenciamento em disco, pois estas são seqüenciais, e, portanto, possíveis de serem organizadas numa árvore B^+. No caso, utilizou-se as estruturas que possuiam melhores desempenhos em aplicações geográficas, segundo [Samet85] e [Shaff89], e, devido a isto, não se preocupou em implementar outros tipos de estruturas.

Outra vantagem em utilizar *quadtrees* é que estas estruturas facilitam as operações de álgebra de conjunto (união e interseção). Os resultados destas operações são processados em tempo relativamente curto em relação a outras abordagens [Samet84]. Além disso, via *quadtrees*, tais operações são implementadas

sem a necessidade de artifícios complexos, podendo-se utilizar algoritmos relativamente simples.

No contexto deste trabalho objetivou-se o desenvolvimento da base geral de um SGBD espacial, deixando-se para desenvolvimentos futuros a implementação de outras potencialidades no GEOCOPPE. Recomenda-se algumas modificações na base do sistema COPPEREL original, visando a facilitar futuros desenvolvimentos. Algumas limitações atuais poderão ser removidas se a arquitetura do sistema for modificada, de modo que o compilador do COPPEREL passe a reconhecer a linguagem LOPEREL*. Para que isto seja possível o COPPEREL tem que incorporar as funções do PCE e todas as chamadas do GDE ao COPPEREL devem ser substituidas por chamadas a máquina virtual do COPPEREL. Com isto, as visas, asserções e procedimentos poderão ser aceitos sobre os dados espaciais.

Outras limitações que restringem o uso do GEOCOPPE são impostas pelo modo atual de entrada e saída dos dados. Por exemplo, os dados de região são fornecidos apenas em formato raster, recomenda-se então modificações para que futuramente o sistema aceite novos tipos de entrada de dados. A apresentação dos dados de uma relação espacial é feita tupla a tupla, através do comando exibir. Obviamente este tipo de saída não é a mais desejável para o usuário.

A título de desenvolvimento inicial foram implementados apenas três comandos de manipulação de dados, pois através deles já é possível formular consultas, permitindo a utilização do GEOCOPPE como SGBD espacial. Como um trabalho futuro deve-se implementar os demais comandos de manipulação.

As árvores B^+, usadas para armazenar as *quadtrees* nas relações do COPPEREL, podem possuir chaves repetidas. As chaves repetidas são necessárias para implementar as *PMR-f quadtrees*. No entanto, a rotina de remoção de chaves retira da árvore uma chave de cada vez. Desta forma, para remover todas as chaves iguais a um determinado valor, que existam dentro da árvore, necessita-se chamar várias vezes a rotina de remoção. Recomenda-se que tal procedimento seja modificado para aumentar a eficiência do sistema.

Referências

[ARC/INFO] "ARC/INFO: an example of a contemporary geographic information system" em Introductory readings in Geographic Information Systems
Taylor & Francis Francis, 1990

[Anten91] Antenucci, John C.; Brown, Kay; Croswell, Peter L.; Kevany, Michael J. e Archer, Hugh
"Geographic Information Systems - A Guide to the Technology"
Van Nostrand Reinhold, 1991

[Auler91] Auler, Patricia; Souza, Jano M. e Ramirez, Milton R.
"geocoppe: Um sgbd Espacial"
Anais do 6ᶜ Simpósio Brasileiro de Banco de Dados, maio de 1991

[Bauer85] Bauer, Michael A.
"Set Operations on Linear Quadtrees"
Computer Vision, Graphics, and Image Processing 29, pp 248-258, 1985

[Burro86] Burrough, P. A.
"Principles of Geographical Information Systems for Land Resourses Assessment"
Clarendon Press, Oxford, 1986

[Cowen90] Cowen, David J.
"GIS versus CAD versus DBMS: What are the diferences?"
Introductory Reading in Geographic Information Systems, Taylor & Francis, 1990

[Clark86] Clarke, K. C.
"Advances in geographic information systems"
Computers, Environment and Urban Systems, 10, pág. 175-184

[Dange90] Dangermond, Jack
"A Classification of Software Components Commonly used in GIS"
Introductory Reading in Geographic Information Systems, Taylor & Francis, 1990, pág. 30-51

[Dyer80] Dyer, Charles R.; Rosenfeld, Azriel e Samet, Hanan
"Region Representation: Boundary Codes from Quadtrees"
Communications of the ACM, março de 1980, vol. 23, nº 3

[Esri] ARC/INFO User Guide

[Falou89] Faloutsos, Christos e Rego, Winston
"Tri-cell: A data structure for Spatial Objects"
Information Systems, vol 14, n° 2, pág. 131-139, 1989

[Gonça90] Gonçalves, Laércio Antonio Castelo Branco
"O Sub-Sistema de Reconstrução do SGBD COPPEREL"
Disertação de Mestrado, Programa de Sistema COPPE/UFRJ", 1987

[Guttm84] Guttman, Antonin
"R-Trees: A Dynamic Index Structure for Spatial Searching"
Proc. ACM SIGMOD 1984, pág. 47-57, June 1984

[Garga82] Gargantini, Irene
"An effective way to represent quadtrees"
Communication of the ACM, dezembro de 1982, vol. 5, n° 12

[Helle90] Heller, Dan
"Xview Programming Manual: An OPEN LOOK toolkit for X11"
O'Reilly & Associates, Inc, 1990

[Knuth73] Knuth, D. E.
"The Art of Computer Programming"
Vol. 3, Sorting and Searching, Addison-Wesley, Reading, MA

[Marbl90] Marble, Duane F.
"Geographic Information Systems: an overview"
Introductory Reading in Geographic Information Systems, Taylor & Francis, 1990, pág. 9-17

[Matto87] Mattoso, M. L. Q.
"A Incorporação de Ferramentas de Apoio aos Usuários do SGBD COPPEREL"
Dissertação de Mestrado, Programa de Sistemas COPPE/UFRJ, 1987

[Newma78] Newman, William M. e Sproull, Robert F.
"Principles of Interactive Computer Graphics"
McGraw-Hill Book Company, 1978, Second Edition

[Pavli92] Pavlidis, M. G.
"Database management for geographic information systems"
Proceedings, National Conference on Energy Resource Management, 1 pág. 255-260

[Peuqu90] Peuquet, Donna J.
"A conceptual framework and comparison of spatial data models"
Introductory Reading in Geographic Information Systems, Taylor & Francis, 1990, pág. 250-285

[Rouss85] Roussopoulos, Nick e Leifker, Daniel
"Direct Spatial Search on Pictorial Databases Using Packed R-trees"
Proceedings ACM SIGMOD, pag.17-31, Maio 1985

[Salzb86] Salzberg, Betty
"Grid File Concurrency"
Information Systems, vol. 11, n° 3, pág. 235-244, 1986

[Samet80] Samet, Hanan
"Region Representation: Quadtrees from Boundary Codes"
Communications of the ACM, março 1980, vol. 23, n° 3

[Samet90b] Samet, Hanan
"Region Representation: Quadtrees from Binary Arrays"
Computer Graphics and Image Processing 13, pág. 88-93, 1980

[Samet82] Samet, Hanan
"Hierarchical Data Structures for Representing Geographical Information"
Report of University of Maryland, Agosto de 1982, TR-1208

[Samet83] Samet, Hanan e Tamminen, Markku
"Computing Geometric Properties of Images represented by Linear Quadtrees"
Report of University of Maryland, 1983, CS-TR-1359

[Samet83b] Samet, Hanan
"Using quadtrees to represent spatial data"
Report of University of Maryland, Maio de 1983, TR-1287

[Samet84] Samet, Hanan; Rosenfeld, Azriel; Shaffer, C. A.; Nelson, R.C.; e Huang, Y.
"Application of Hierarchical Data Structures to Geographical Information Systems (phase III)
Report of University of Maryland, 1985, ETL-0376

[Samet84b] Samet, Hanan
"The Quadtree and Related Hierachical Data Structure"
Computer Surveys, vol. 16, n°, junho de 1984

[Samet85] Samet, Hanan e Rosenfeld, Azriel
"Application of hierarchical data structures to geographical information Systems (phase IV)
Report of University of Maryland, 1985, TR 1578

[Samet85b] Samet, Hanan e Shaffer, Clifford A.
"Using linear quadtrees to store vector data"
Report of University of Maryland, 1985, TR 1550

[Samet89a] Samet, Hanan
"The Design and Analysis of Spatial Data Structures"
Addison Wesley, 1989

[Samet89b] Samet, Hanan
"Aplications Spatial Data Structures"
Addison Wesley, 1989

[Shaff87] Shaffer, Clifford. A. e Samet, Hanan.
"Optimal Quadtree Construction Algorithms"
Computer Vision, graphics, and Image Processing 37, 1987, pág. 42-419

[Shaff89] Shaffer, Clifford. A.; Samet, Hanan e Nelson, Randal C.
"QUILT: A Geographic Information System based on Quadtrees"
Report of University of Maryland CSS-TR-1885.1, Fevereiro 1989

[Selli87] Sellin, Timos; Roussopoulos, Nick e Faloutsos, Christos
"The R^+-Tree: A Dynamic Index for Multi-Dimensional Objects"
Proceedings of the 13th VLDB Conferenc, Brighton 1987, pág. 507-518

[Smith87] Smith, P. D. e Barnes, G. M.
"File & Databases: an Introduction"
Addison-Wesley Publishing Company, 1987

[Tomli90] Tomlinson, Roger F.
"Geographic Information Systems - a new frontier"
Introductory Reading in Geographic Information Systems, Taylor & Francis, 1990, pág. 18-29

[Trott87] Trotta, Claudio Newton Ferreira
"Extensões no SGBD Copperel para aplicações não convencionais"
Dissertação de Mestrado, Programa de Sistemas COPPE/UFRJ, 1989

[Zakim85] Zakimi, B. M.; Mattoso, M. L. Q. e Prazeres, M. J.
"Manual do Usuário do LOPEREL (Linguagem de Operação do SGBD COPPEREL)"
Relatório Técnico ES-62/85, COPPE/UFRJ, 1985