UNIVERSIDADE DO EXTREMO SUL CATARINENSE – UNESC
CURSO DE CIÊNCIA DA COMPUTAÇÃO

WAGNER COMIN SONÁGLIO

ANÁLISE COMPARATIVA DO DESEMPENHO DOS MÉTODOS DE ACESSO PARA DADOS ESPACIAIS: ESTUDO DE CASO COM O POSTGIS

CRICIÚMA, JULHO DE 2007

WAGNER COMIN SONÁGLIO

ANÁLISE COMPARATIVA DO DESEMPENHO DOS MÉTODOS DE ACESSO PARA DADOS ESPACIAIS: ESTUDO DE CASO COM O POSTGIS

Projeto de Pesquisa do Trabalho de Conclusão do Curso de Ciência da Computação da Universidade do Extremo Sul Catarinense.

Orientador: Prof. M.Sc. Daniel Pezzi da Cunha
Co-orientadora: Profa. M.Sc. Lucimar Fátima Siqueira

CRICIÚMA, JULHO DE 2007

Aos meus pais, Ulisses Sonáglio e Libera Comin Sonáglio, meus irmãos, e a minha namorada Sabrina Gonçalves Rogério.

AGRADECIMENTOS

Agradeço, primeiramente a Deus, por estar sempre comigo em todos os momentos de minha vida. Agradeço de uma forma geral, a todos aqueles que contribuíram de alguma maneira para que este trabalho pudesse ser realizado. A todos, meus mais sinceros agradecimentos.

“Não adies o momento da batalha, nem espera que tuas armas se enferrujem e o fio de tuas espadas se embote. A vitória é o principal objetivo da guerra”

(Sun Tzu)

RESUMO

O banco de dados geográficos é um importante componente dos Sistemas de Informações Geográficas, é nele onde todos os dados espaciais são armazenados para uma eventual consulta. Para melhorar a eficiência destas consultas utilizam-se os Métodos de Acesso Espaciais. O objetivo principal deste trabalho é apresentar uma análise comparativa de desempenho entre os Métodos de Acesso Espaciais chamados GiST e B-Tree, utilizando como ferramenta o Sistema Gerenciador de Bancos de Dados Espaciais PostGIS e uma aplicação desenvolvida em Java para a realização dos testes.

Palavras chave: Sistemas de Informações Geográficas, Bancos de Dados Geográficos, Métodos de Acesso Espacial.

ABSTRACT

The geographic database is an important component of the Geographic Information Systems, it´s where the spatial data is saved for eventual consultation. To improve the efficiency of these consultations Spatial Access Methods are used. The main goal of this project is to present a perfomance comparing analysis between the Spatial Access Methods GiST and B-tree, using as a tool the Spatial Database Manager PostGIS and an application developed with Java for the carrying of the tests.

Keywords: Geographic Information System, Geographic Databases, Spatial Access Methods.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interface*
CPU	*Central Processing Unit*
CVS	*Concurrent Version System*
GEOS	*Geometry Engine Open Source*
GPS	*Global Positioning System*
GUI	*Graphical User Interface*
HTTP	*HyperText Transfer Protocol*
J2EE	*Java 2 Enterprise Edition*
J2ME	*Java 2 Micro Edition*
J2RE	*Java 2 Runtime Environment*
J2SE	*Java 2 Standard Edition*
JDBC	*Java Data Base Connectivity*
JDK	*Java Development Kit*
JSF	*Java Server Faces*
JTS	*Java Topology Suite*
MAE	Métodos de Acesso Espaciais
MAM	Métodos de Acesso Multidimensionais
MRE	Menor Retângulo Envolvente
MVCC	*MultiVersion Concurrency Control*
ODBC	*Open Data Base Connectivity*
OGC	*Open Geospatial Consortium*
PSQL	*Postgres Structured Query Language*

RAD	*Rapid Application Development*
RAM	*Random Access Memory*
SFSSQL	*Simple Features Specification for SQL*
SGBD	Sistema Gerenciador de Bancos de Dados
SGBDG	Sistema Gerenciador de Bancos de Dados Geográficos
SIG	Sistemas de Informações Geográficas
SQL	*Structured Query Language*
SRID	*Spatial Referecing Identifier*
SRS	*Spatial Referecing System*
UFRJ	Universidade Federal do Rio de Janeiro
UVI	Unidade Visualizadora de Imagens
WKB	*Well-Known Binary*
WKT	*Well-Know Text*

SUMÁRIO

1 INTRODUÇÃO

O uso de computadores na manipulação de grandes quantidades e variedades de dados tem levado ao desenvolvimento dos chamados sistemas de informação, dedicados ao armazenamento e análise integrada dos dados.

Segundo Brito e Rosa (1996), os Sistemas de Informações Geográficas (SIG) são um caso específico de sistema de informação. Um SIG pode ser definido como um sistema destinado à aquisição, armazenamento, manipulação, análise e apresentação de dados referidos espacialmente na superfície terrestre.

Há quase duas décadas, os bancos de dados tornaram-se o componente central dos sistemas de informação. Assim como os sistemas de informação, os SIG adotam como ponto central da sua arquitetura os bancos de dados, só que neste caso mais específico são adotados os bancos de dados geográficos. Um banco de dados geográficos armazena e recupera dados espaciais em diferentes geometrias (imagens, vetores, grades) e informações descritivas (atributos não-espaciais) armazenadas em tabelas (CÂMARA et al, 2005).

Em um sistema de loteamentos, por exemplo, todos os lotes e logradouros de uma cidade estão cadastrados em uma base de dados geográficos e isso gera uma grande quantidade de informações. As consultas espaciais nesta base de dados serão ineficientes se não forem otimizadas. Tendo em vista este problema surgiram os Métodos de Acessos Espaciais (MAE). O principal objetivo de um MAE é propiciar uma rápida obtenção dos objetos espaciais que satisfazem um determinado relacionamento topológico, métrico ou direcional projetando um caminho otimizado aos dados (CIFERRI, 2000). O espaço é indexado de tal forma que, por exemplo, a recuperação dos objetos espaciais contidos em uma área pequena requeira apenas o acesso aos objetos próximos a esta área em oposição à análise do conjunto completo de objetos armazenados em memória secundária. Segundo Câmara et al

(2005), existem vários métodos de acesso espacial, sendo os mais tradicionais: B-Trees, k-d trees, grid files, quad-trees e R-trees. Para verificar qual o MAE mais eficiente no processamento de consultas em um banco de dados geográficos, será feito no projeto, um estudo comparativo de dois tipos diferentes de MAE chamados GiST e B-Tree, utilizando como ferramenta o Sistema Gerenciador de Banco de Dados Geográficos (SGBDG) PostGIS, que é uma extensão gratuita para armazenamento e tratamento de dados espaciais do Sistema Gerenciador de Banco de Dados (SGBD) PostgreSQL.

1.1 OBJETIVO GERAL

Estudar os bancos de dados geográficos e realizar uma análise comparativa do desempenho de seus métodos de acesso aos dados espaciais utilizando o PostGIS, um SGBDG.

1.2 OBJETIVOS ESPECÍFICOS

Os objetivos específicos desta pesquisa são:

a) realizar uma pesquisa sobre os bancos de dados geográficos;
b) documentar e estudar o PostGIS;
c) pesquisar e adquirir bases de dados geográficos para a realização de testes;
d) desenvolver uma aplicação para acessar as bases de dados geográficos e utilizar algoritmos de desempenho para a realização do teste de eficiência dos métodos de acesso espaciais;
e) disponibilizar uma análise comparativa do desempenho dos métodos de acessos espaciais.

1.3 JUSTIFICATIVA

Segundo Câmara et al (2005) os bancos de dados geográficos possuem um grande volume de informações e consultas a sua base de dados espaciais sem a utilização de uma metodologia geralmente são ineficientes. Os MAE são estruturas de dados auxiliares, mas que são muito importantes para a eficiência no processamento de consultas espaciais. Normalmente uma consulta espacial envolve apenas uma pequena parcela do banco de dados, porque percorrer todo o banco pode ser ineficiente. Portanto um plano de execução para a consulta tipicamente começa com uma fase de filtragem, que identifica quais objetos podem satisfazer a qualificação da consulta.

Durante toda a pesquisa, será documentada informações referentes aos fundamentos de bancos de dados geográficos e do PostGIS, a extensão espacial do sistema gerenciador de banco de dados PostgreSQL e informações comparativas sobre os métodos de acesso espaciais. A análise do desempenho destes métodos definirá qual o mais eficiente para as consultas aos bancos de dados geográficos. Este desempenho será analisado a partir de uma aplicação gráfica criada, que trabalhará integrada com o banco de dados espaciais desenvolvido. O desempenho destes métodos será avaliado por meio de uma aplicação de software que usará algoritmos para calcular a eficiência dos acessos ao banco. O PostGIS foi escolhido para a realização dos testes porque é a extensão para tratamento de dados espaciais do SGDB chamado PostgreSQL.

1.4 ESTRUTURA DO TRABALHO

Este projeto é formado por oito seções. Na seção 1 há uma introdução aos objetivos do projeto, os temas envolvidos e a justificativa para a realização do mesmo.

Nas seções 2, 3 e 4 são abordados os temas relacionados a SIG, bancos de dados geográficos e MAE.

A seção 5 apresenta trabalhos correlatos onde o objetivo dos mesmos é semelhante ao deste trabalho.

As ferramentas, métodos e dados utilizados para a realização dos testes de desempenho, bem como as suas especificações e problemas encontrados são abordados na seção 6.

A seção 7 apresenta o resultado do desempenho dos MAE, de acordo com os resultados obtidos nas etapas anteriores.

Por fim, há uma conclusão e as possibilidades para trabalhos futuros.

As tabelas com os resultados obtidos nos testes seguem como apêndice.

2 SISTEMAS DE INFORMAÇÕES GEOGRÁFICAS

O termo Sistema de Informação é usado como sinônimo de sistemas de base de dados ou sistema de processamento de dados. Essencialmente ele refere-se a um sistema embora nem sempre computadorizado, desenhado para entrada, armazenamento, processamento e saída de informações. O produto final, a informação, normalmente será o resultado da composição de entidades, construídos a partir de um ou vários dados. Dentro deste contexto, o sistema de informação dá ao usuário a habilidade de produzir informações, especialmente contribuindo na redução de incertezas (PAREDES, 1994). Sendo assim pode-se considerar que a função de um sistema de informação é prover informação ao usuário de modo a executar ou adotar decisões na pesquisa, no planejamento e no gerenciamento.

Um Sistema de Informação pode ser visto como um subsistema auxiliar do sistema do mundo real. É auxiliar porque sua existência é condicionada às necessidades de previsão e intercâmbio de informação dentro de um grande sistema.

Com a constante evolução da tecnologia da informação, surgiram os SIGs. Estes sistemas são usualmente aceitos como sendo uma tecnologia que possui o ferramental necessário para realizar análises com dados espaciais e, portanto, oferecem, ao serem implementados, alternativas para o entendimento da ocupação e utilização do meio físico, compondo o chamado universo da Geotecnologia, ao lado do Processamento Digital de Imagens e da Geoestatística.

2.1 CONCEITOS BÁSICOS DE UM SIG

Um SIG é um sistema de informação baseado em um computador que permite capturar, modelar, manipular, recuperar, consultar, analisar e apresentar dados que possuem uma referência no espaço (CÂMARA et al, 2005). A tecnologia de SIG pode trazer enormes benefícios devido à sua capacidade de manipular a informação espacial de forma precisa, rápida e sofisticada (GOODCHILD 1985 apud PAREDES, 1993).

Como sistema considera-se um arranjo de entidades (elementos) relacionadas ou conectadas, de tal forma que constituem uma unidade ou um todo organizado, com características próprias e subordinadas a processos de transformação conhecidos. As entidades são os elementos ou objetos tomados como unidades básicas para a coleta de dados. Os dados relacionam-se com os atributos, que caracterizam e fornecem significado à unidade estudada (BRITO; ROSA, 1996). Por exemplo, um “terreno de uma cidade” pode ser uma entidade e as suas características de solo, revelo e uso da terra seus atributos.

Como informação geográfica considera-se o conjunto de dados cujo significado contém associações ou relações de natureza espacial, dados esses que podem ser apresentados em forma gráfica (pontos, linhas e, áreas/polígonos), numérica e alfanumérica. Assim, um SIG utiliza uma base de dados computadorizada que contém informação espacial sobre a qual atuam uma série de operadores espaciais (TEIXEIRA; CHRISTOFOLETTI, 1992).

Os SIGs incluem-se no ambiente tecnológico chamado de geoprocessamento, cuja área de atuação envolve a coleta e tratamento da informação espacial, assim como o desenvolvimento de novas aplicações e sistemas.

A tecnologia ligada ao geoprocessamento envolve hardware (equipamentos) e software (aplicativos), que são destinados à implementação de sistemas para fins didáticos, de

pesquisa acadêmica ou aplicações profissionais e científicas nos mais diversos ramos das ciências (TEIXEIRA; CHRISTOFOLETTI, 1992).

Em um SIG a apresentação de dados tem papel relevante na extração de informações. Ela é usada para visualizar o problema, possibilitando observar, manipular e estudar os relacionamentos geográficos envolvidos e também pode apresentar alternativas à solução do problema considerado (EGENHOFER 1990 apud CÂMARA et al, 2005). Tais técnicas têm avançado significantemente e sua importância tem estabelecido papel relevante para o gerenciamento de recursos.

2.2 COMPONENTES DE UM SIG

Um sistema de informação geográfica é composto por três componentes, que necessitam ser balanceados para o funcionamento do sistema: hardware e sistema operacional, software de aplicação (o SIG em si) e aspectos institucionais do SIG.

2.2.1 Hardware e Sistema Operacional

De uma maneira geral, os componentes de hardware de um SIG são apresentados de acordo com a Figura 1.

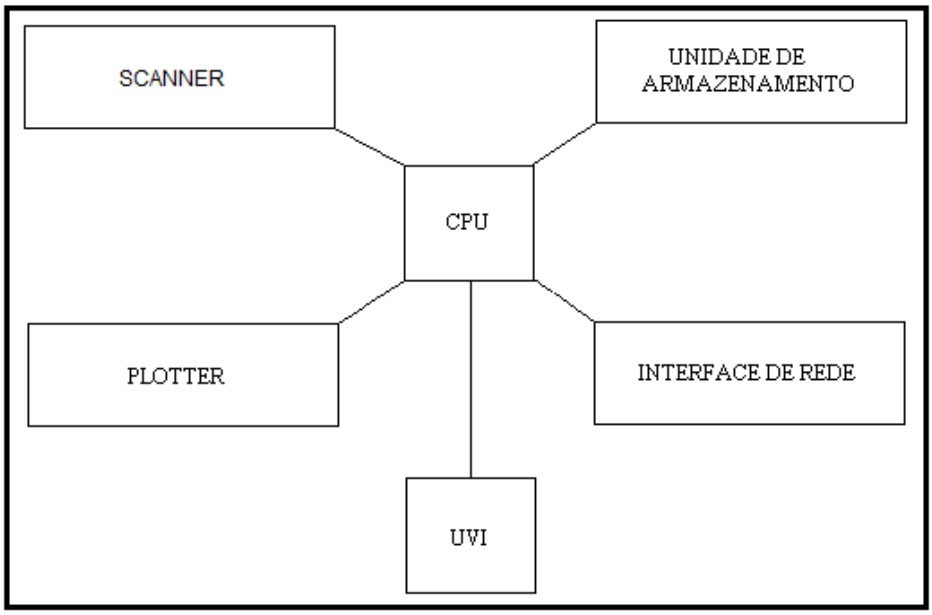

Figura 1. Componentes de Hardware de um SIG
Fonte: Adaptado de BRITO, J.; ROSA, R. (1996)

O computador ou Unidade Central de Processamento (CPU) é ligado a uma unidade de armazenamento, onde é feito o armazenamento dos dados. O scanner ou outro dispositivo de entrada similar é usado para converter os dados analógicos (mapas, cartas, entre outros) em dados digitais. Uma unidade visualizadora de imagens (UVI), impressora ou plotter são usados para visualizar o processamento efetuado em cima dos dados. Uma interface de rede é usada para comunicação com outros sistemas ou computadores para troca de informações ou processamento compartilhado.

2.2.2 Software de Aplicação (SIG)

Um sistema de informação geográfica é composto de forma simplificada por cinco componentes ou subsistemas:

a) **subsistema de entrada de dados (aquisição)**: realiza a conversão dos dados analógicos para dados digitais. Dados estes provenientes de fotos aéreas, imagens de satélite, cartas, mapas, GPS, entre outras fontes. Estas informações podem ser inseridas por meio de um scanner, teclado, entre outros;

b) **subsistema de gerenciamento de dados (banco de dados)**: consiste na utilização de um SGDB para realizar a inserção, remoção, atualização ou modificação dos dados. Um banco de dados espacial armazena e recupera os dados espaciais e alfanuméricos de um SIG;

c) **subsistema de análise e manipulação de dados**: consiste na análise de dados que tenha informações de interesse, a fim gerar novas informações. Podem ser realizadas tarefas de seleção e/ou agregação de informações estatísticas e/ou temáticas;

d) **subsistema de saída e apresentação dos dados**: relacionado a apresentação dos resultados na fase de entrada e/ou análise e manipulação dos dados. Os resultados podem ser tabelas, relatórios, documentos, cartas, imagens exibidos em monitores de vídeo, gráficos, internet, entre outros.

2.2.3 Aspectos Institucionais de um SIG

Os cinco subsistemas exibem o processamento dos dados em um SIG, mas isto não quer dizer que um determinado SIG será usado realmente. Para o funcionamento efetivo de um SIG é necessário um lugar apropriado no contexto institucional e de pessoas com qualificação para a utilização do sistema. Existem diversas formas de aplicação: acadêmica, pesquisa, empresas, entre outros. Grandes investimentos são necessários, tanto na aquisição de hardware/software quanto na qualificação do pessoal.

A escolha e implantação de um SIG é uma tarefa de médio a longo prazo (BRITO; ROSA, 1996). A melhor maneira de implantar um SIG é por meio das seguintes etapas:

a) **identificação das necessidades do usuário:** é a tarefa mais difícil na escolha e implantação de um SIG. O usuário deve identificar os problemas a serem resolvidos no seu universo de atuação;

b) **levantamento detalhado da instituição:** consiste no levantamento do nível de informatização da instituição, softwares, hardware, qualificação do pessoal e base de dados existente;

c) **detalhamento dos produtos necessários:** fase muito importante na definição do sistema. Devem-se especificar os produtos geográficos a serem produzidos e sua necessidade de ligação com um banco de dados;

d) **escolha do sistema de geoprocessamento:** o SIG deve ser composto por programas de alto nível, ser genérico e preencher as necessidades do projeto. Tem de operar em sistemas multi-usuário, multitarefa e permitir a ligação com um SGBD. Deve-se considerar também o custo do sistema, suporte técnico e pessoal qualificado;

e) **execução de um projeto piloto:** consiste na experimentação do sistema escolhido, ou seja, é a etapa onde o sistema é testado pelos usuários;

f) **implantação do sistema propriamente dito:** etapa onde o sistema encontra-se operacional, pronto para executar todas as funções especificadas na primeira fase.

2.3 DADOS ESPACIAIS OU GEOGRÁFICOS

A geografia e seus dados descritos fazem parte do cotidiano da humanidade em geral. Geralmente cada decisão tomada pelas pessoas é influenciada ou ditada pela geografia (PAREDES, 1994).

Desta forma, informação geográfica é o conjunto de dados (físicos, sociais, econômicos, entre outros), cujo significado contém uma associação ou relação com uma localidade específica (KUBO 1985 apud PAREDES, 1994).

O mundo real consiste de muitas características geográficas. Elas permitem realizar diversas análises e responder perguntas como: onde está? Quais são suas características? A geografia é vista na base de dados do SIG como um depósito composto por objetos, cujo espaço é definido pela relação entre estes objetos.

2.3.1 Propriedades dos Dados Geográficos

Todos os dados e fenômenos possuem propriedades que precisam ser conhecidas especialmente para a geocodificação. Geralmente, são as seguintes propriedades (PAREDES, 1994):

a) **localização**: usam-se as duas dimensões (X,Y) para determinar a localização no plano terrestre e a terceira dimensão (Z) para determinar a elevação da superfície. Estas são as propriedades básicas para determinar a localização. Dessa forma a localização é uma característica fundamental dos dados geográficos;

b) **volumetria**: bases de dados geográficos contêm muitas informações e muitos dos problemas de processamento geográfico estão ligados ao grande conjunto de dados. Por isso o armazenamento de dados é uma característica importante;

c) **dimensionamento**: a cartografia divide as entidades em pontos, linhas e áreas, o mensuramento varia desde nominal, ordinal, intervalo até razão;

d) **continuidade**: a continuidade é uma propriedade geográfica importante, mas ela nem sempre obedece à distribuição estatística. Porém a classificação do espaço por áreas ou regiões é possível graças a continuidade;

e) **tamanho**: muitos fenômenos geográficos podem ser facilmente medidos se suas base de dados estiver codificada. Elas são implementadas usando-se certos algoritmos;

f) **distribuição**: a densidade é uma das medidas da distribuição de fenômenos geográficos no espaço. A densidade é calculada pelos atributos e pela contagem dos objetos cartográficos em um conjunto de entidades geográficas;

g) **padrão**: os padrões descrevem a estrutura da distribuição dos fenômenos geográficos;

h) **vizinhança**: enquanto o padrão é a repetição de um atributo no espaço, a vizinhança define a variação do dado geográfico no espaço;

i) **contigüidade**: está relacionado à justaposição dos dados, é uma expressão geográfica do campo topológico. É expressa de diversas maneiras e definida em termos de limite partilhado, como redes, indicando conectividade;

j) **forma**: representa a composição dos gráficos de pequenas dimensões. Sua medição direta é muito difícil e complexa;

k) **escala**: é uma propriedade quantitativa dos dados cuja representação varia e sua faixa é limitada pela finalidade cartográfica e dos fenômenos. Promove precisão e característica métrica aos dados geográficos.

2.3.2 Tipos de Dados Geográficos

Os dados geográficos compreendem três tipos (PAREDES, 1994):

a) **dado espacial**: se refere à localização, à forma e às relações entre as entidades espaciais;

b) **dado descritivo**: são as características da entidade espacial;

c) **dado temporal**: caracteriza o período ou época da ocorrência do fenômeno ou fato geográfico.

Numa forma mais simplificada o SIG aborda dois tipos de dados principais: os geométricos que descrevem características do próprio espaço e os não-geométricos que descrevem outros tipos de características. Os dados geométricos se caracterizam por serem:

a) **posicionais**: caracterizam a posição do objeto no espaço;

b) **topológicas**: referem-se aos relacionamentos de vizinhança ou conexão entre os objetos;

c) **amostrais**: representam valores de grandezas físicas de um ponto ou região.

Os dados não-geométricos descrevem características e informações auxiliares de natureza não geométrica. Para estes tipos de dados podem ser usados bancos de dados convencionais, ao invés dos bancos de dados geográficos.

2.3.2.1 Características dos Dados Espaciais

Os dados espaciais são aqueles que referenciam no espaço terrestre por meio de coordenadas “x,y” (latitude, longitude) que definem a sua localização e por uma coordenada “z” que representa o atributo elevação (altitude).

Os dados espaciais descrevem objetos do mundo real em termos de posição a sistema de coordenadas, de atributos e outros parâmetros espaciais e descrevem a união desses dados entre si (Figura 2). Assim a:

a) **localização**: define a condição geométrica da entidade;

b) **relação**: define a topologia das entidades;

c) **descrição**: define os atributos das entidades.

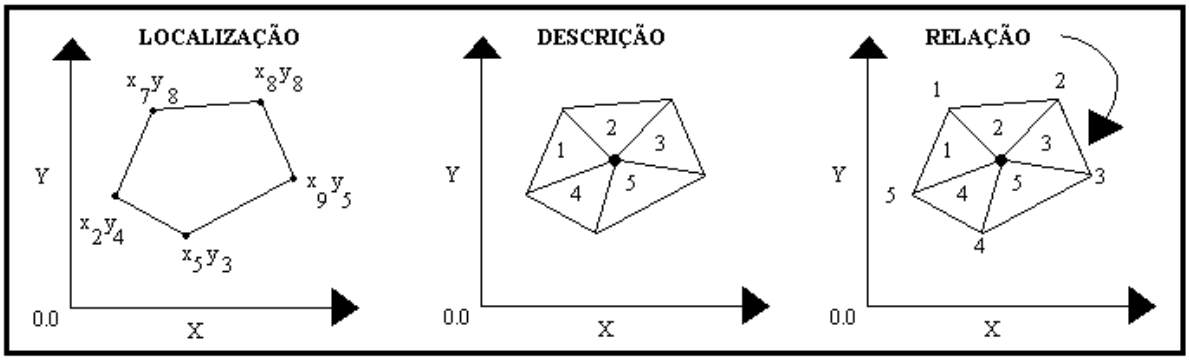

Figura 2. Exemplo de localização, descrição e relação
Fonte: Adaptado de PAREDES, E. (1994)

Existem conjuntos de definições dos objetos espaciais, os quais estão divididos em classes baseadas na dimensão espacial dos mesmos. Estes objetos são (PAREDES, 1994):

a) **pontos**: um objeto zero dimensional que especifica uma localização geométrica. Abrangem todas as entidades geográficas, pois são perfeitamente posicionadas por um único par de coordenadas “x,y” (latitude, longitude);

b) **linhas**: um objeto um dimensional é uma linha entre dois pontos, seqüência de pontos com coordenadas diferentes, coordenada inicial e final;

c) **polígonos**: Um objeto dois dimensional que pode apresentar diversas formas e descrevem as propriedades topológicas de áreas, como por exemplo forma, vizinhança, entre outros.

A Figura 3 exibe as variações e condições destes tipos de dados:

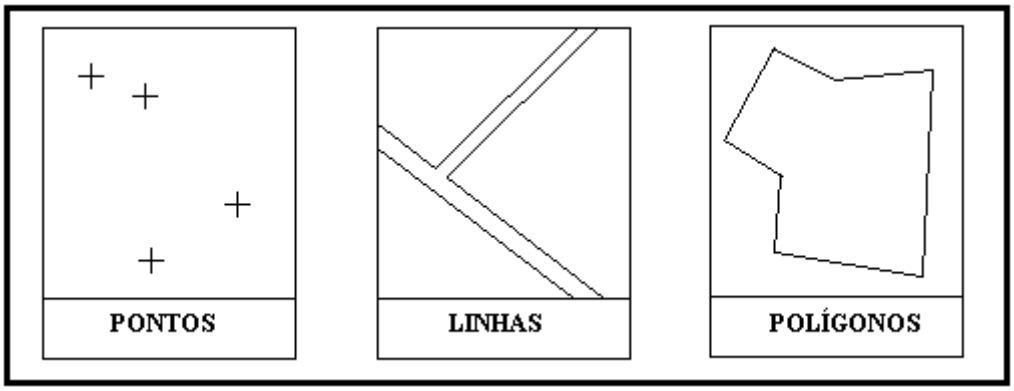

Figura 3. Classes de objetos espaciais
Fonte: Adaptado de PAREDES, E. (1994)

2.3.2.2 Características dos Dados Descritivos

Os dados descritivos são formados por itens de informações e estatísticas referentes a aspectos físicos, ambientais, sócio-ecônomicos, infra-estrutura, entre outros. Alguns exemplos:

a) **físico-ambientais**: características físico-geográficas, estruturas, uso do solo, topografia;

b) **sociais**: referentes a informações como: habitação, saúde, educação, população, segurança;

c) **econômicas**: informações relacionadas a empregos, renda;

d) **infra-estrutura**: informações relacionadas aos serviços urbanos e rurais, sistemas viários, energia.

A Figura 4 ilustra as variações e condições destes tipos de dados:

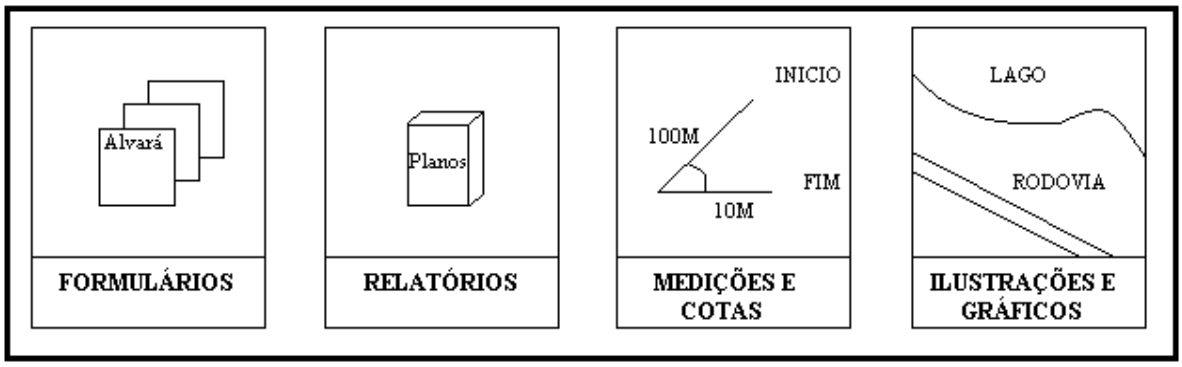

Figura 4. Características dos dados descritivos
Fonte: Adaptado de PAREDES, E. (1994)

2.3.2.3 Características dos Dados Temporais

Os dados geográficos possuem atributos temporais e espaciais (Figura 5) como as dimensões de um cubo. A dimensão "v" corresponde ao objeto em si e sua capacidade de abranger a faixa de valores de acordo com escala de medição e as dimensões "t" e "s" representa os intervalos temporais próprios e as unidades espaciais, respectivamente.

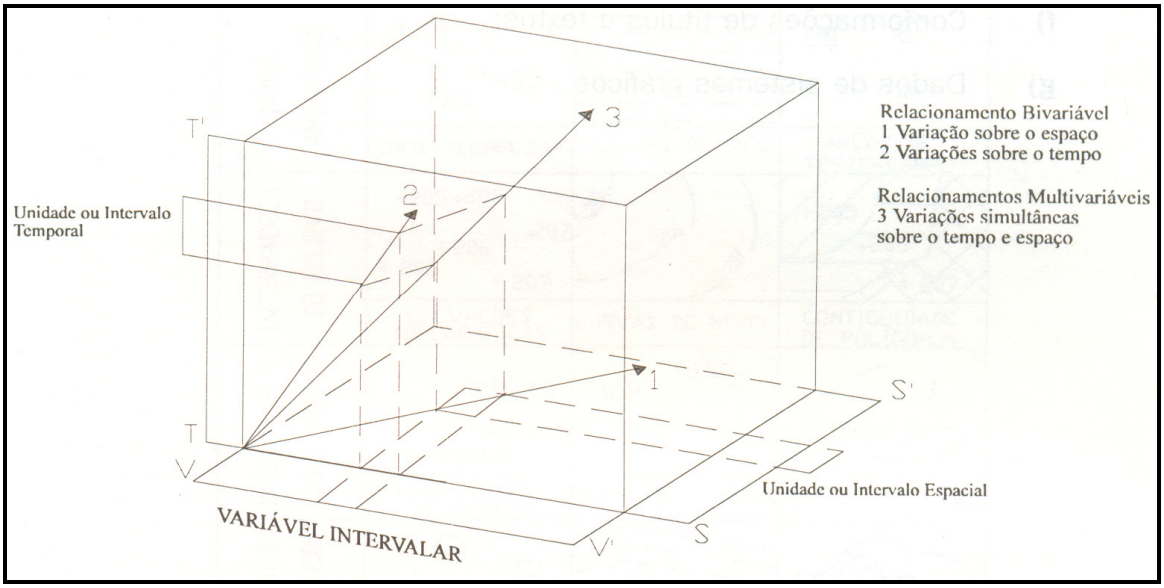

Figura 5. Atributos temporais e espaciais de dados geográficos
Fonte: PAREDES, E. (1994)

2.4 MODELAGEM DE DADOS ESPACIAS

O aspecto cognitivo é um fator importante na percepção espacial. No modelo humano de percepção espacial, os conceitos usados para compreender o espaço são freqüentemente baseados em noções que necessitam de uma definição formal, sem serem implementados diretamente (MAEDA; SALES; SIMONATO, 2005). Ou seja, necessitam de um modelo conceitual para representar as informações, como apresentado na Figura 6.

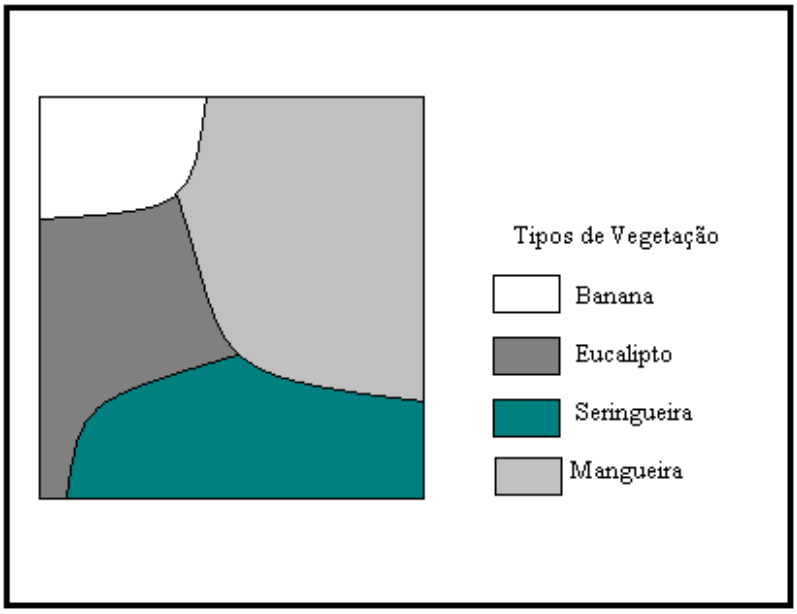

Figura 6. Representação de um modelo de campos.
Fonte: Adaptada de MAEDA, V.; SALES, R.; SIMONATO, T. (2005)

Em um SIG, a abstração na modelagem de dados espaciais é ainda mais importante. Os sistemas geográficos necessitam de maior riqueza de detalhes. Existe também a necessidade de incluir na conceituação, aspectos da representação gráfica que esse pretende adotar para as entidades: campos ou objetos, vetores ou imagens, sem contar os aspectos estéticos, de visualização. Alguns dos conceitos mais importantes para representação de dados espaciais estão descritos a seguir (MAEDA; SALES; SIMONATO, 2005):

a) **modelo de campos**: também conhecido como visão de campos, define o mundo real sendo visto como uma superfície continua onde as entidades

espaciais podem variar continuamente. Em um mapa de vegetação, por exemplo, cada ponto representa um tipo de vegetação, como visto na Figura 6;

b) **modelo de objetos**: também chamado de visão de espaços, define o mundo real contendo entidades identificáveis no espaço. Cada entidade é identificada como sendo um dado individual, com atributos que o distinguem dos demais. Um exemplo deste modelo está na Figura 7.

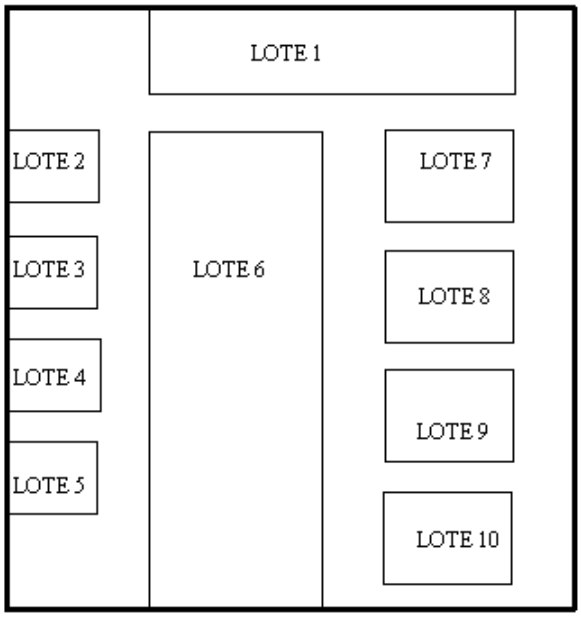

Figura 7. Representação de um modelo de objetos
Fonte: Adaptado de MAEDA, V.; SALES, R.; SIMONATO, T. (2005)

Tipos de representações espaciais:

a) **representação matricial ou *raster*** (Figura 8): primeiro e mais antigo formato de dados, o formato matricial ou *raster* divide toda a área de estudo que está sendo representada em grades regulares, constituindo células (pixels). Cada célula contém um valor único e é individualmente integrada ao sistema por suas coordenadas. Torna-se mais simples, modelar o espaço assim representado como uma matriz p(i,j), composto de i linhas e j colunas. Este espaço quando preenchido, define a localização das entidades na área de estudo. Dados

matriciais são facilmente manipuláveis computacionalmente, seu problema é o grande requerimento de espaço de armazenamento;

b) **representação vetorial** (Figura 9): usa entidades como pontos e linhas (coordenadas x,y) para identificar as localizações. Tenta reproduzir um elemento o mais exatamente possível, com um grau de precisão muito maior. Os métodos vetoriais assumem que as coordenadas dos pontos são matematicamente exatas e permitem que dados complexos sejam armazenados em menos espaço do que o método matricial, porém seu problema é que seus cálculos são mais complexos e exigem mais tempo de raciocínio.

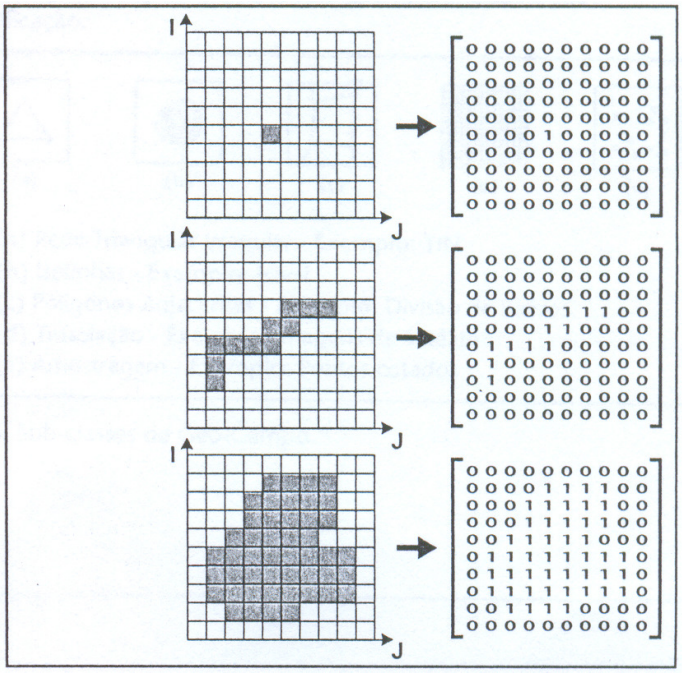

Figura 8. Representação matricial
Fonte: MAEDA, V.; SALES, R.; SIMONATO, T. (2005)

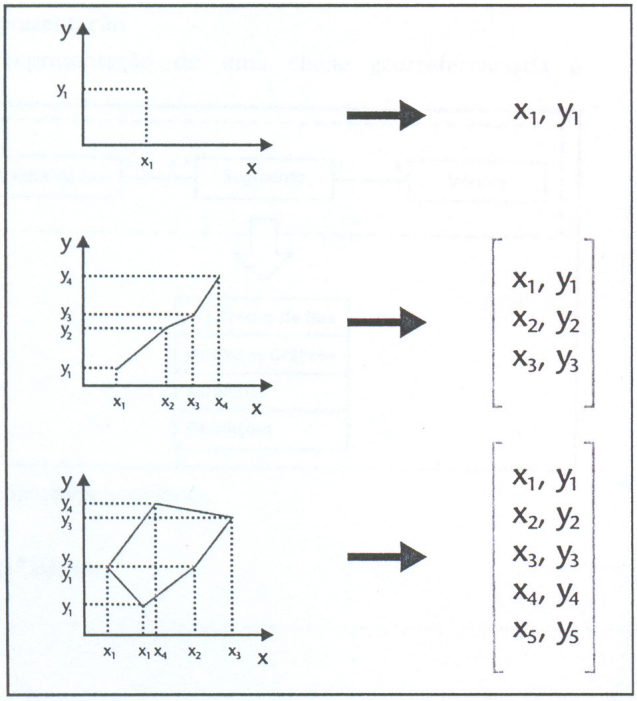

Figura 9. Representação vetorial
Fonte: MAEDA, V.; SALES, R.; SIMONATO, T. (2005)

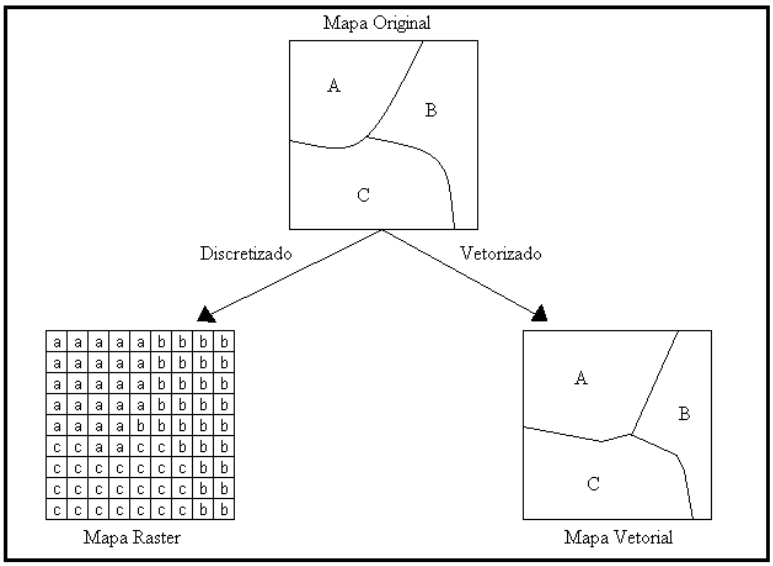

Figura 10. Comparação entre a representação *raster* e vetorial
Fonte: Adaptado de PAREDES, E. (1994)

3 BANCOS DE DADOS

Segundo Câmara et al (2005), há quase duas décadas os bancos de dados tornaram-se o componente central de sistemas de informação, tanto do ponto de vista de projeto, quanto do ponto de vista de operação. Esta evolução foi possível graças a uma sólida tecnologia desenvolvida para armazenamento e manipulação de dados convencionais.

O principal objetivo de um banco de dados é prover uma visão abstrata dos dados ocultando do usuário os detalhes de como os dados são armazenados e mantidos. Os dados, normalmente, apresentam uma grande complexidade e esta complexidade é ocultada por meio dos níveis de abstração.

Os bancos de dados contam com várias operações que facilitam a pesquisa e a manipulação dos dados. Trata-se de um conjunto de tabelas inter-relacionadas com capacidade de controle dos dados cadastrados. Estes dados poderão ser acessados e manipulados mediante rotinas de controle e segurança previamente estabelecidos no seu desenvolvimento.

3.1 SISTEMA GERENCIADOR DE BANCO DE DADOS

Um SGBD é o conjunto de programas de computador (softwares) responsáveis pelo gerenciamento de uma base de dados. O seu papel é retirar da aplicação cliente a responsabilidade de gerenciar o acesso, manipulação e organização dos dados. O SGBD disponibiliza uma interface para que os seus aplicativos clientes possam incluir, alterar, ou consultar dados.

Segundo Silva (1999), o principal objetivo de um SGDB é proporcionar um ambiente conveniente e eficiente para retirar, armazenar e atualizar as informações contidas

no banco de dados. Os principais desafios que os SGDBs enfrentam são: redundância de dados, inconsistência, dificuldade no acesso aos dados, isolamento de dados, múltiplos usuários, segurança e integridade dos dados.

A redundância refere-se a presença de dados repetidos no banco. A inconsistência consiste em várias cópias de dados que apresentam diferenças entre si. A dificuldade no acesso aos dados ocorre devido a complexos "caminhos" de acesso aos dados. O isolamento de dados é a inexistência de caminhos entre os dados semelhantes. Múltiplos usuários podem estar acessando o banco e estes dados precisam estar disponíveis em tempo real. A segurança é um item importante, pois nenhum usuário pode apagar, alterar, ou transferir dados que estejam fora de sua permissão. A integridade dos dados consiste na inexistência de dados que não representem os fenômenos do mundo real.

3.2 RELACIONAMENTOS

Para que se possa ter uma visão geral sobre sistemas de banco de dados é necessário que alguns conceitos sejam bem estabelecidos. Segundo Korth e Silberschatz (1994 apud SILVA, 1999, p. 148), entidade corresponde a um objeto que existe e é perfeitamente distinguível de outros objetos.

Conjuntos de entidades correspondem a um grupo de entidades do mesmo tipo. Relacionamento refere-se a uma associação existente entre várias entidades. Um conjunto de relacionamentos é um grupo de relacionamentos do mesmo tipo. Cardinalidade de mapeamento representa o número de entidades ao qual outra entidade pode estar associada por meio de um relacionamento. Para um conjunto de relacionamentos binários entre dois tipos de conjuntos, existem os diferentes tipos de cardinalidades de mapeamento: um-para-um

(Figura 11), um-para-muitos (Figura 11), muitos-para-um (Figura 12) e muitos-para-muitos (Figura 12).

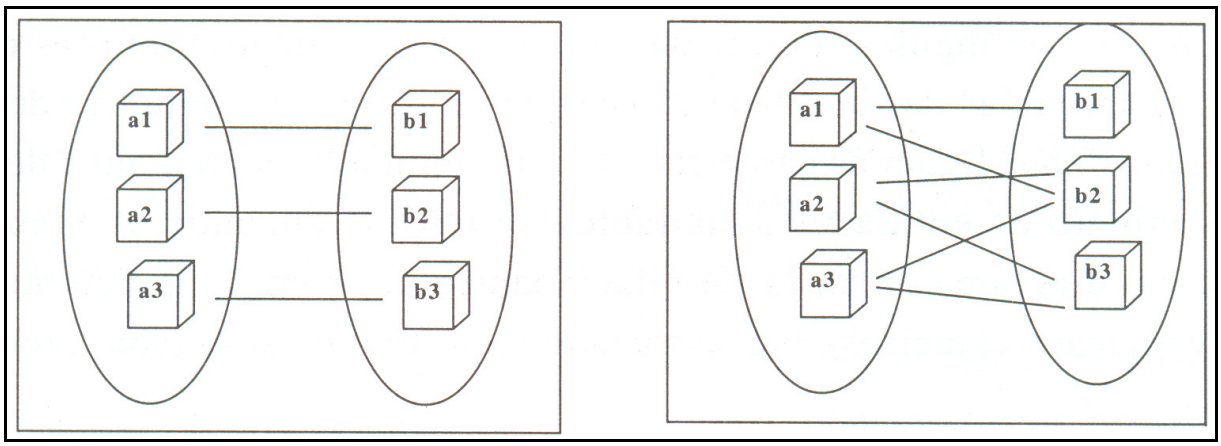

Figura 11. Relacionamento um-para-um e um-para-muitos
Fonte: SILVA, A. (1999)

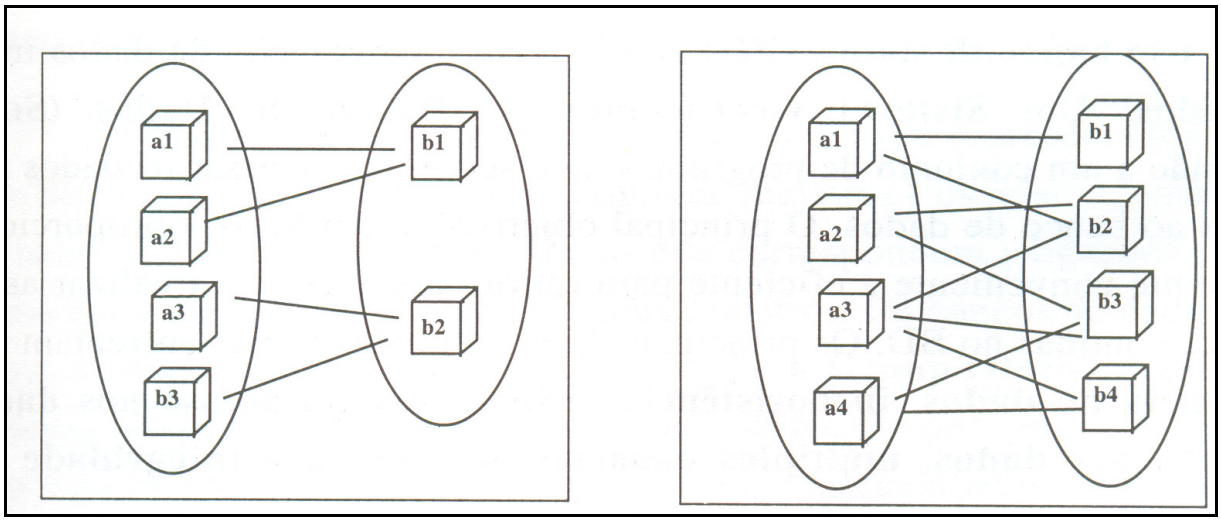

Figura 12. Relacionamento muitos-para-um e muitos-para-muitos
Fonte: SILVA, A. (1999)

O relacionamento um-para-um corresponde a quando uma entidade em um conjunto está associada a apenas uma entidade em outro conjunto. O relacionamento um-para-muitos é definido quando uma entidade de um conjunto está associada a qualquer número de entidades de outro conjunto. No relacionamento muitos-para-um qualquer número de entidades em um conjunto pode estar associado a apenas uma entidade de outro conjunto. No relacionamento muitos-para-muitos qualquer número de entidades de um conjunto pode estar associada a qualquer número de entidades de outro conjunto.

3.3 BANCOS DE DADOS GEOGRÁFICOS

Para dar suporte ao armazenamento e recuperação de dados dos SIGs, surgiram os bancos de dados geográficos. Na Figura 13 observa-se que o SIG é responsável pela parte de processamento dos dados geográficos, visualização e integração enquanto o banco de dados geográficos armazena e recupera os dados espaciais.

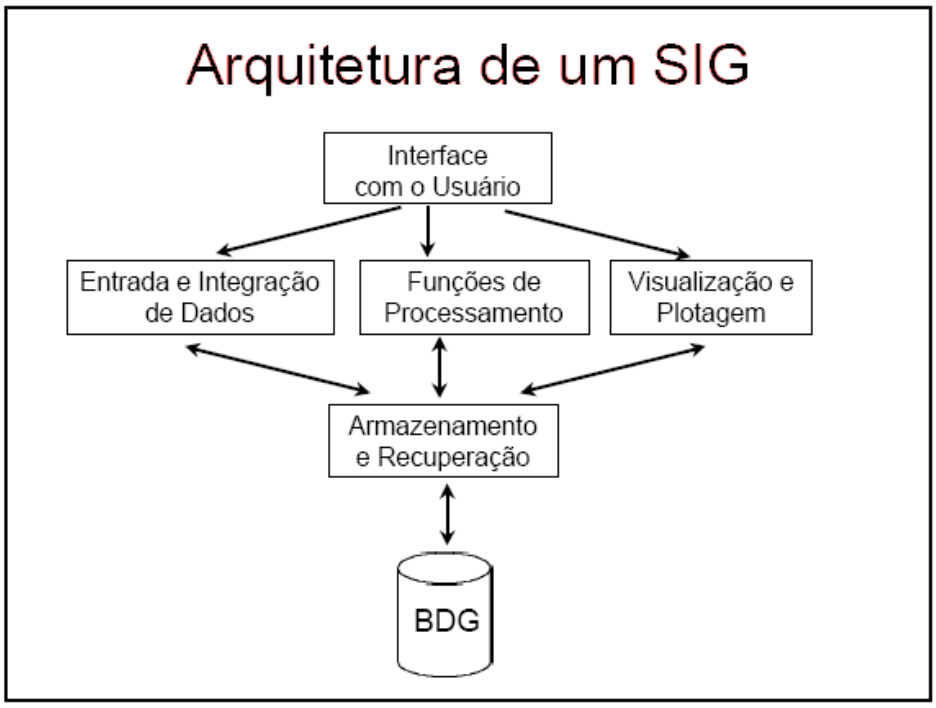

Figura 13. Posição de um banco de dados geográficos na arquitetura de um SIG
Fonte: MELLO, R. (2004)

Um banco de dados convencional possui apenas atributos alfanuméricos que descrevem suas características (atributos convencionais), os bancos de dados geográficos possuem além destes atributos, atributos espaciais que descrevem sua forma, indicam sua localização no espaço e a sua validade. É um repositório de dados do mundo real que são geo-referenciados.

O principal diferencial em relação aos bancos de dados convencionais é a análise geo-espacial, definido como o conjunto de funções aplicadas sobre um objeto espacial como validação dos dados de entrada, verificação da existência de relacionamento topológico,

visualização de mapas e busca de informações geométricas/estatísticas. As modificações de componentes convencionais e temporais e espaciais podem ser feitos por meio de comandos do banco de dados. Além dos mesmos tipos de consulta dos bancos de dados convencionais, o geográfico também possui critérios de consultas espaciais. Eles podem apresentar diversas formas de visualização, como 2D e 3D, combinação de resultados de consultas, visualização simultânea de vários objetos, apresentação de dados estatísticos e operações especiais, como zoom, translação, rotação, entre outros.

4 MÉTODOS DE ACESSO ESPACIAIS

Um MAE, também conhecido na literatura como Método de Acesso Multidimensional (MAM), é uma estrutura de indexação voltada ao suporte de objetos espaciais, especialmente de retângulos (CIFERRI; SALGADO, 2001).

Segundo Câmara et al (2005), MAE, ou índices espaciais, são estruturas de dados auxiliares, mas essenciais para o processamento eficiente de consultas espaciais. De fato, normalmente uma consulta espacial envolve apenas uma pequena parcela do banco de dados. Neste caso, pesquisar o banco inteiro pode ser ineficiente. Portanto, um plano de execução para a consulta começa com uma etapa de filtragem, que identifica quais objetos podem satisfazer a qualificação da consulta. Esta fase depende da existência de índices espaciais, em geral definidos sobre aproximações de geometrias dos objetos.

O espaço é indexado de tal forma que, por exemplo, a recuperação de objetos espaciais contidos em um espaço qualquer, requeira apenas o acesso aos objetos próximos a este espaço, sem a necessidade de utilizar objetos armazenados em uma memória secundária. Portanto, um MAE é projetado como um caminho otimizado aos dados e o seu uso aumenta o desempenho de SGBDGs no processamento de consultas.

Uma forma comum de indexar um conjunto de dados é por meio das árvores de pesquisa. Estas estruturas permitem que algumas operações, como a localização de um elemento, sejam executadas em tempo logarítmico – $O(\log n)$. Na Figura 14 têm-se um exemplo de árvore binária balanceada, que é empregada para uso em memória principal, onde as cidades estão organizadas por ordem alfabética.

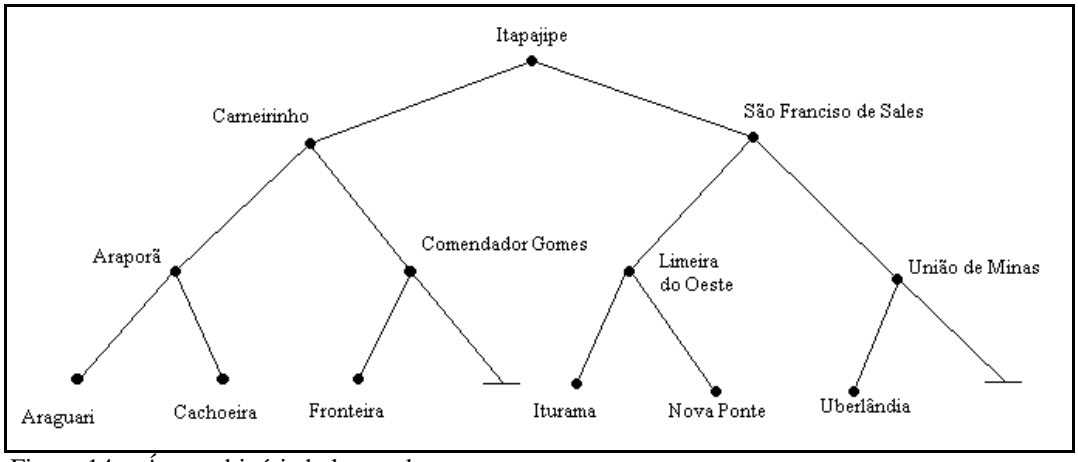

Figura 14. Árvore binária balanceada
Fonte: Adaptado de CÂMARA, G. et al (2005)

Em grandes bancos de dados onde à memória principal não pode ser suficiente para armazenar todos os nós da árvore, é comum armazená-los em disco. Nestes casos, é necessária uma representação que minimize o acesso a disco. Segundo Câmara et al (2005), a forma mais comum e largamente empregada pelos sistemas comerciais atuais, é a representação do índice por meio de uma árvore B^+, que tenta minimizar o acesso a disco durante a etapa de consulta agrupando várias chaves (Figura 15) em um único nó, denominando de página.

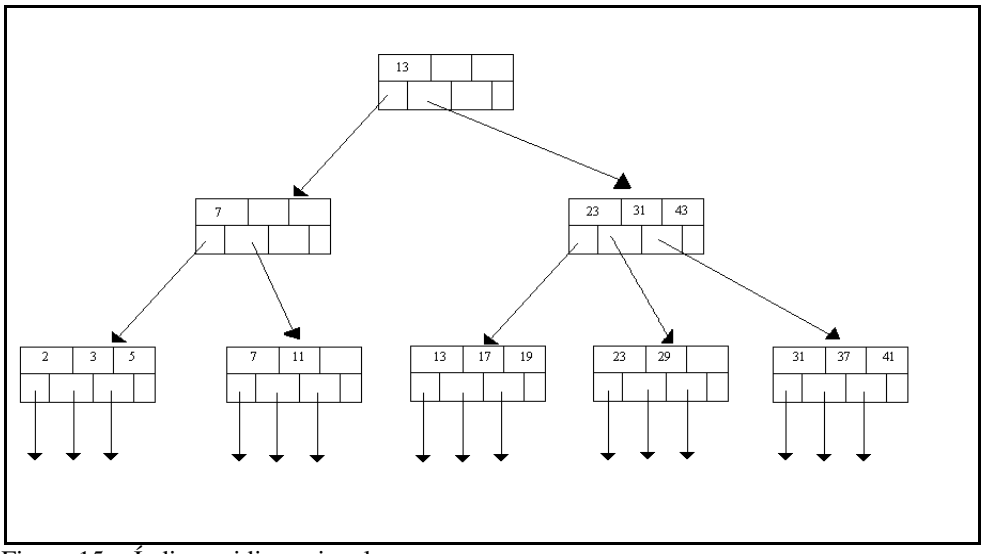

Figura 15. Índice unidimensional
Fonte: Adaptado de CÂMARA, G. et al (2005)

Tanto a árvore binária quanto a B^+ são estruturas unidimensionais, ou seja, elas pressupõem que a chave de pesquisa seja formada por apenas um atributo ou pela

concatenação de vários atributos. Em sistemas que trabalham com informações multidimensionais (mais de um atributo), como os sistemas de bancos de espaciais, estas estruturas não são diretamente aplicáveis. Para estes sistemas existem os MAM, que estão ligados ao processamento de consultas de dados espaciais.

Um conceito importante dos métodos de acesso espaciais é uso de aproximações, ou seja, a estrutura do índice trabalha com representações mais simples dos objetos, como o Menor Retângulo Envolvente (MRE) do objeto. Nesta forma, o processo de consulta é divido em duas fases: uma de filtragem e outra de refinamento (Figura 16). Na fase de filtragem são usados métodos de acesso para reduzir os candidatos que satisfaçam a consulta. Esta etapa é importante, pois na próxima fase, o refinamento, ocorre a aplicação de algoritmos geométricos computacionalmente complexos e custosos e que é aplicada a geometria exata dos candidatos selecionados na etapa anterior.

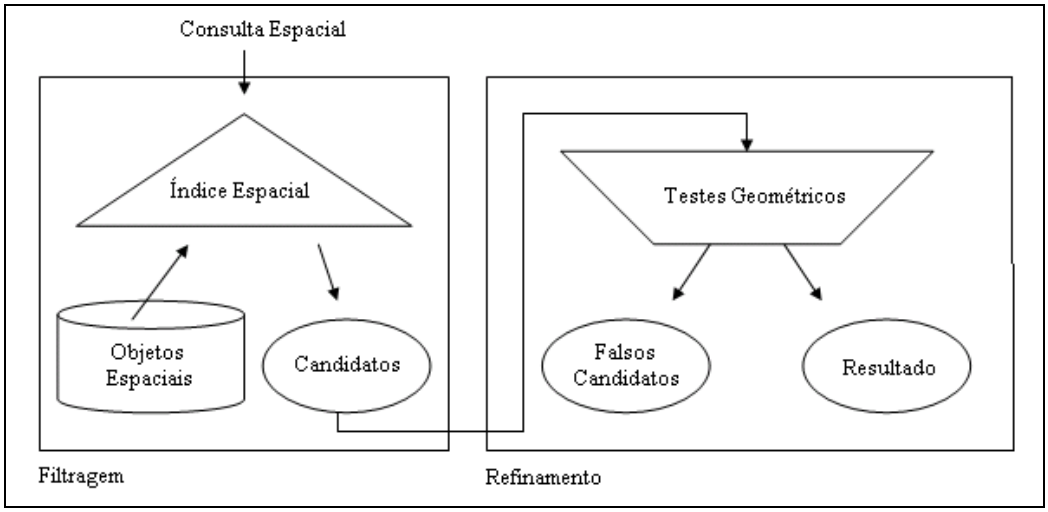

Figura 16. Processamento de consultas espaciais
Fonte: Adaptado de CÂMARA, G. et al (2005)

4.1 B-TREE

Criada em 1972 por Bayer e MacCreight para solucionar o problema das árvores binárias, onde a inserção ou remoção de dados exigia que toda a árvore fosse reorganizada. Na B-Tree as informações estão armazenadas em forma de nós que possuem uma "semelhança" com suas ramificações. Segundo Paradeda (2002), este método de indexação é uma árvore *n*-ária, isto é, cada nó pode ter *n* filhos, sendo que em geral este valor é escolhido de tal maneira a otimizar a blocagem física do arquivo de índices. Para manter estes tamanhos pré-definidos, os nós internos podem ser unidos ou divididos. Para inserir ou remover variáveis de um nó, o nó não poderá ultrapassar sua ordem e nem ser menor que sua ordem dividida por dois.

O método de indexação B-Tree é segundo Ooi e Tan (2002) uma das estruturas de dados mais usadas e estudadas. É uma árvore balanceada, de multi-caminhos e a organização externa de arquivos, desde a raiz até a folha, têm o mesmo tamanho. Todo nó não-folha (exceto a raiz) na árvore tem [*n*/2] e *n* filhos, onde *n* (chamado de ordem) é fixada para a árvore em particular. Ela mantém a sua eficiência a despeito de inserção e remoção de dados e uma alta escalabidade, como por exemplo, se o tamanho do banco de dados for incrementado por um fator de 100-400, será necessário acrescentar apenas um nível na sua estrutura.

4.1.1 Características da família B-Tree

Uma B-tree de ordem n é uma árvore de pesquisa de multi-caminhos com as seguintes propriedades:

a) A raiz terá no mínimo duas sub-árvores, a menos que esta seja uma folha;

b) todo nó não-raiz e não-folha terá $k - 1$ chaves e k ponteiros para as sub-árvores onde $[n/2] \leq k \leq n$;

c) todo nó folha terá $k - 1$ chaves onde $[n/2] \leq k \leq n$;

d) todos os nós folha serão do mesmo nível;

e) um nó com k sub-árvores armazena $k - 1$ chaves ou registros;

f) todos os nós de derivação (aqueles que não são folhas) possuem exclusivamente sub-árvores não vazias.

Sendo assim, uma B-tree é sempre pelo menos "parcialmente completa", tem muitos níveis e é perfeitamente balanceada (OOI; TAN, 2002). Tipicamente todo nó corresponde a uma página do disco. A Figura 17 (a) mostra uma simples estrutura de B-Tree, aonde os valores das 18 chaves vão de 6 até 89.

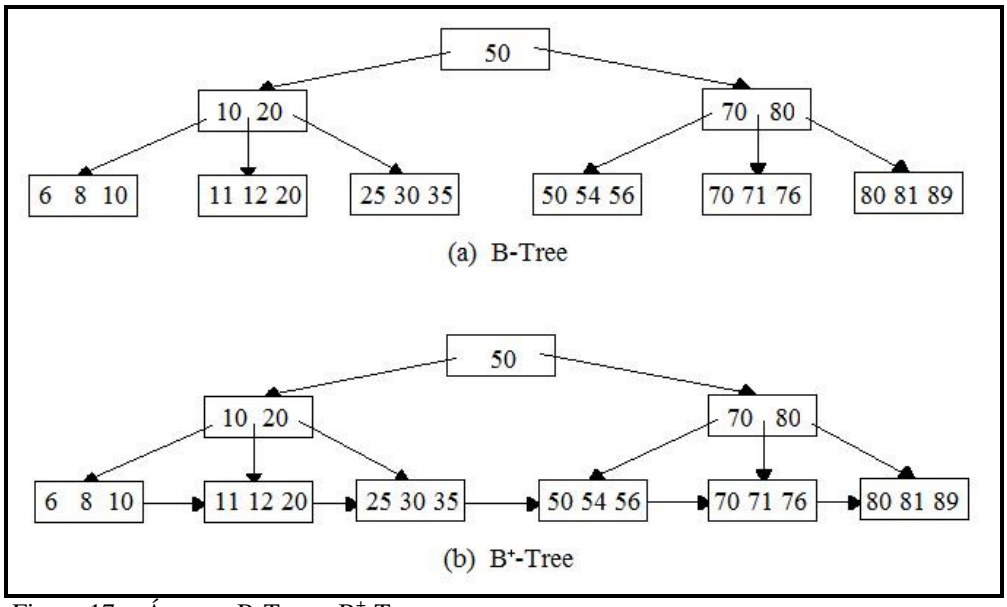

Figura 17. Árvores B-Tree e B^+-Tree
Fonte: OOI, B. C.; TAN, K. (2002)

Desde que cada nó de uma B-Tree represente uma página, acessar um nó significa acessar apenas uma vez o disco. Conseqüentemente quanto menor o número nós, melhor. Para uma melhor utilização da capacidade de armazenamento dos nós B-Tree, foi criada uma variante chamada B^*-Tree. Neste método, todos os nós exceto a raiz são requeridos para ser pelo menos "2/3 completa", não apenas "parcialmente completa" como as B-Trees.

Enquanto as estruturas da B-Tree providenciam uma rápida e exata busca dos termos (número de acessos ao disco é igual à altura da árvore mais uma página de dados), buscas de escala requerem uma "profundidade primeiramente transversal" da árvore que irá fazer a verificação de alguns nós internos várias vezes. Para reduzir o custo operacional é indicado uso das B^+-Trees. B^+-Tree se diferencia de B-Tree de várias formas. Primeiro, todos os valores de chave que aparecem nos nós-folha. Os valores dentro dos nós internos não precisam ser chaves, eles servem apenas de separadores para direcionar o processo de busca. Como é mostrado na Figura 17 (b), todas as 18 chaves estão armazenadas dentro dos nós-folha e os valores 22, 59 e 78 são separadores que não aparecem no banco de dados. Segundo, todos os nós folha estão conectados (seguindo o próximo ponteiro folha), enquanto um valor maior que 6 é encontrado.

B-Trees podem ter muitos filhos, sendo assim, o fator de ramificação desse tipo de árvore pode ser muito grande, sendo determinado pelas características de disco. Uma B-Tree pode ser usada na implementação de um conjunto de operações dinâmicas em tempo 0 (log n). Os números de acessos ao disco requeridos pela maioria das operações de uma B-Tree são proporcionais a sua altura (NONATO, 2004).

Para analisar a altura n no pior caso da árvore, será adotado um teorema onde se $n \geq 1$ então para todo elemento n-key de uma árvore de altura h e grau mínimo $t \geq 2$, $h \leq \log t$ $n+1/2$.

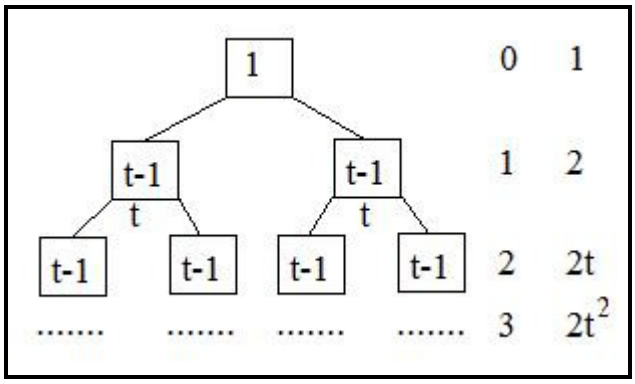

Figura 18. Altura de uma árvore B-Tree
Fonte: Adaptado de PARADEDA, R. B. (2002)

Se uma B-Tree tem altura *h,* seu número de nós é minimizado quando a raiz contém um elemento e todos outros nós contém *t* - 1 elementos. Neste caso, há 2 nós com profundidade 1, 2t nós com profundidade 2, $2t^2$ nós com profundidade 3 e assim por diante, até a profundidade *h* ter $2t^h$-1 nós. Na Figura 18 pode-se ver uma árvore com altura $h = 3$.

Demonstrando o teorema:

$n \geq 1 + (t - 1)\sum 2t^{i-1}$

$= 1 + 2(t - 1)(t^H - 1/t - 1)$

$= 2t^h - 1.$

Uma B-Tree é de ordem *n* quando *n* representa o número máximo de filhos que um nó pode ter (exceto a raiz). Por exemplo, uma árvore de ordem 8 pode ter no máximo 16 filhos e no mínimo 8. Alguns autores utilizam a palavra “ordem” para indiciar o número máximo de chaves de um nó.

4.1.2 Operações Básicas

A principal característica das B-Trees está nos seus métodos de inserção e remoção que sempre deixam a árvore balanceada. No caso de árvores binárias, inserções aleatórias podem deixar a árvore desbalanceada (COMER, 1979). Enquanto uma árvore não

balanceada como mostrada na Figura 19 (a) tem longos e também pequenos caminhos, uma árvore balanceada exibida na Figura 19 (b) deixa tudo no mesmo nível.

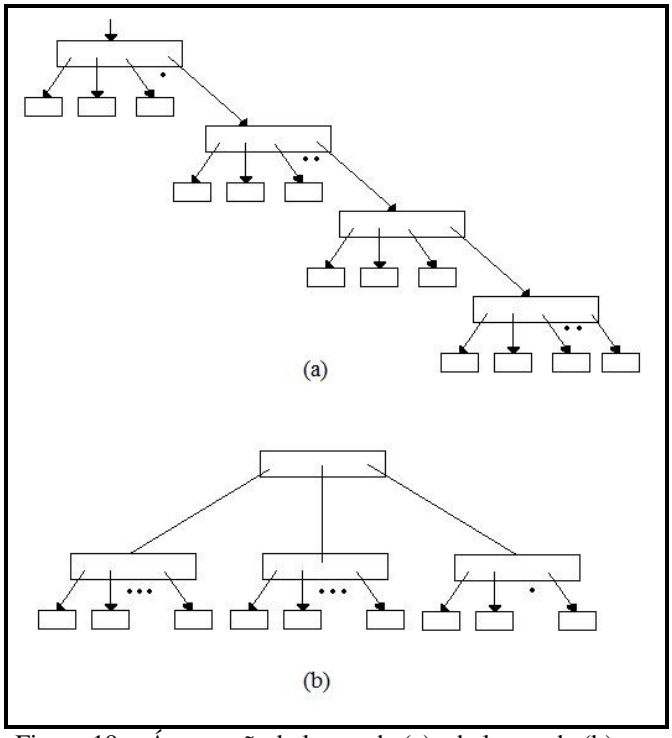

Figura 19. Árvore não-balanceada (a) e balanceada (b).
Fonte: Extraído de COMER, D. (1979).

O caminho mais longo em uma B-Tree de *n* chaves contém no máximo algo como $\log_d n$ nós, *d* sendo a ordem da árvore como exibido na forma da Figura 20. Uma operação de busca pode passar por todos os nós em uma árvore não-balanceada que indexa um arquivo de registro *n*, mas nunca passa mais do que $1 + \log_d n$ nós em uma B-Tree de ordem *d* para qualquer arquivo. Porque cada visita requer apenas um acesso secundário de armazenamento e equilibrando a árvore reduz-se o custo de processamento. O esquema de balanceamento da B-Tree restringe mudanças na árvore de um único caminho de uma folha para a raiz, assim não se pode introduzir um "nó fugitivo" que cause queda de desempenho. Além disso, o mecanismo de balanceamento usa armazenamento extra para diminuir o custo de balanceamento.

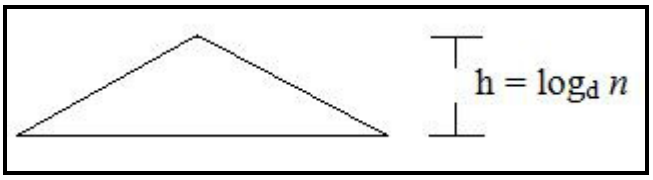

Figura 20. Forma de uma B-Tree de ordem *d* indexando um arquivo de *n* registros
Fonte: Adaptado de COMER, D. (1979)

4.1.2.1 Buscas

A operação de busca em uma B-Tree é semelhante à de uma árvore binária, onde a busca por um valor específico começa examinando a raiz. Se o valor for igual à raiz, o valor existe na árvore. Se o valor for menor do que a raiz então deve-se buscar na sub-árvore da esquerda e assim recursivamente em todos os nós da sub-árvore. Similarmente, se o valor for maior do que a raiz então deve buscar na sub-árvore da direita. Até alcançar o nó folha da árvore, encontrando ou não o valor requerido.

Em vez de escolher entre um nó-filho a direita ou a esquerda como em uma árvore binária, a busca tem que fazer uma escolha de *n*-caminhos. O nó-filho correto é escolhido executando uma procura linear nos valores do nó. Após achar um valor maior ou igual ao valor desejado, o nó-filho aponta ao valor seguido a esquerda. Se todos os valores forem menores que o valor desejado, o nó-filho mais a direita é seguido. A procura pode também ser terminada assim que o nó desejado for encontrado. O tempo corrente da operação de procura depende da altura da árvore, que é $O(\log_t n)$.

4.1.2.2 Inserção

Para ver como o balanceamento é mantido na inserção, observa-se a Figura 21 (a) que mostra uma B-Tree de ordem 2. Desde que cada nó de ordem *d* contenha entre *d* e 2*d*

chaves, todo nó na figura tem entre 2 e 4 chaves. Alguns indicadores não são descritos, mas devem estar presentes em cada nó para marcar o número atual de chaves. A inserção de uma nova chave requer um processo de dois passos. Primeiro, uma busca procederá da raiz até o local da folha para inserção. Então, a inserção é executada e o balanceamento é refeito por meio de um procedimento que parte da folha para trás (até a raiz). Observando a Figura 21 (a), um pode ser visto quando é inserida a chave "57" e a busca termina sem sucesso na quarta folha. Desde que a folha possa armazenar outra chave, a nova chave é simplesmente inserida, rendendo a árvore mostrada na Figura 21 (b).

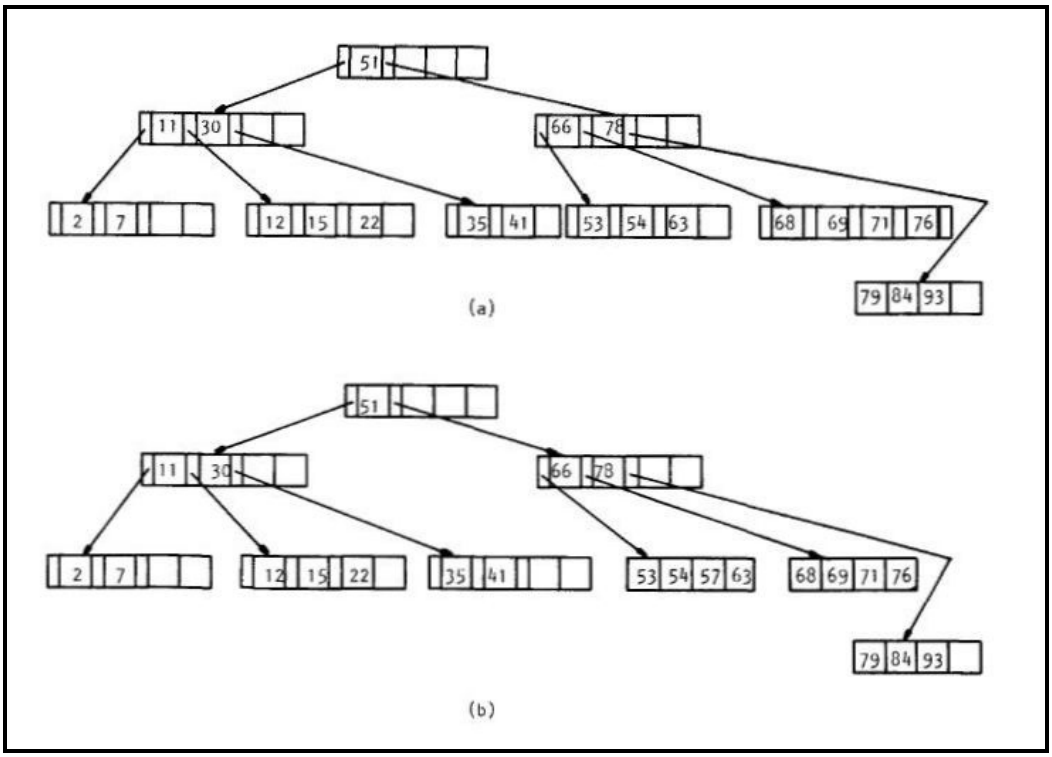

Figura 21. Uma B-Tree de ordem 2 (a), e após a inserção do valor "57"(b)
Fonte: COMER, D. (1979)

Se a chave "72" for inserida, as complicações irão aparecer, já que a folha apropriada está cheia. Sempre que uma chave precisa ser inserida em nó que já esteja cheio, uma função de divisão (*Split*) ocorrerá, o nó é dividido como é exibido na Figura 22. Das $2d + 1$ chaves, a menor chave está localizada em um nó e a maior chave está armazenada em outro nó e o valor restante é colocado em um nó-pai que serve como um separador. Usualmente, o nó-pai irá armazenar um nó adicional e a inserção irá terminar. Se o nó-pai estiver cheio, o

mesmo processo de divisão será feito novamente. Neste caso, dividir propagará todo o caminho até a raiz e aumentará a altura da árvore em um nível por causa da divisão da raiz (COMER, 1979).

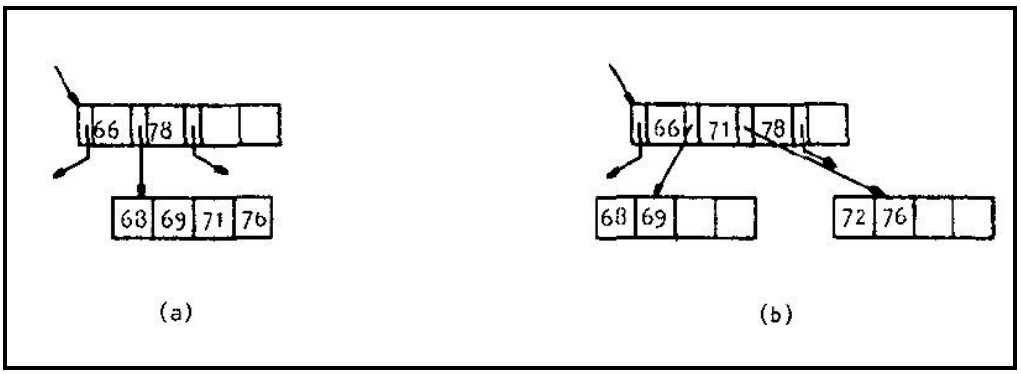

Figura 22. Uma folha e seu ancestral em uma B-Tree (a). A mesma árvore depois da inserção do valor "72" (b)
Fonte: COMER, D. (1979)

4.1.2.3 Remoção

A remoção em uma B-tree também requer uma operação de busca que irá localizar o nó a ser apagado. Aqui há duas possibilidades: o nó a ser apagado está em uma folha, ou está em um nó não-folha. A remoção de um nó não-folha requer que uma chave adjacente seja alocada até a posição vazia para que funcione corretamente. Para localizar uma chave em uma ordem seqüencial de chaves, somente uma busca pela folha mais a esquerda na sub-árvore direita da vaga agora vazia pode ser feita. Em uma árvore de busca binária, o valor procurado reside sempre em uma folha. A Figura 23 demonstra estes relacionamentos, onde a remoção do valor "17" requer que a próxima chave seqüencial, "21" seja removida e colocada em uma posição vazia. A próxima chave seqüencial sempre ficará na folha mais a esquerda da sub-árvore dada pelo ponteiro direito da posição vazia.

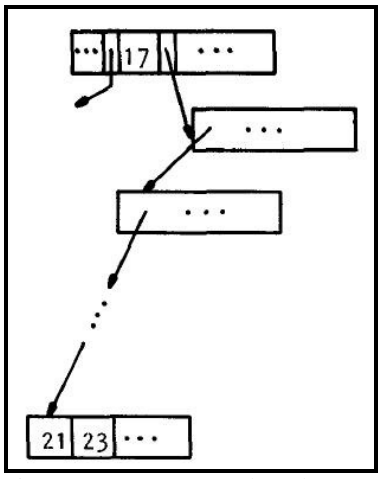

Figura 23. Remoção do valor “17
Fonte: COMER, D. (1979)

Quando o espaço vazio for movido para uma folha, precisam-se checar quais chaves *d* permaneceram. Se for menos que *d* chaves ocupando a folha, uma tentativa de leitura de um espaço vazio irá ocorrer (*underflow*), e a redistribuição dessas chaves será necessária. Para re-balancear, somente uma chave é requerida, que pode ser emprestada de uma folha vizinha. Mas desde que a operação requeira no mínimo dois acessos ao armazenamento secundário, uma melhor redistribuição dividirá uniformemente as chaves remanescentes entre os dois nós relacionados, diminuindo o custo das sucessivas remoções no mesmo nó. A redistribuição está ilustrada na Figura 24.

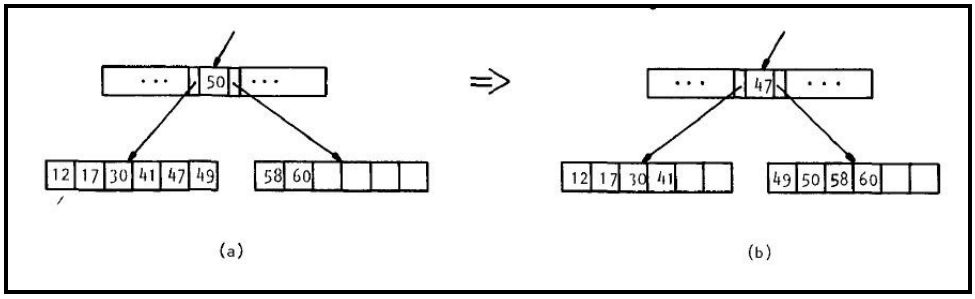

Figura 24. Parte de uma B-Tree antes (a) e depois (b) da redistribuição
Fonte: COMER, D. (1979)

Uma distribuição de chaves entre dois vizinhos será satisfatória apenas se ter no mínimo 2*d* chaves para distribuir. Quando menos do que 2*d* valores remanescerem, uma concatenação ocorrerá. Durante a concatenação, as chaves são simplesmente combinadas em um dos nós e o outro é descartado. Desde que um nó permaneça, a separação de nós no

antecessor não será mais necessária, isso também está adicionado à única folha remanescente. A figura 25 exibe a concatenação e a localização final da chave separadora.

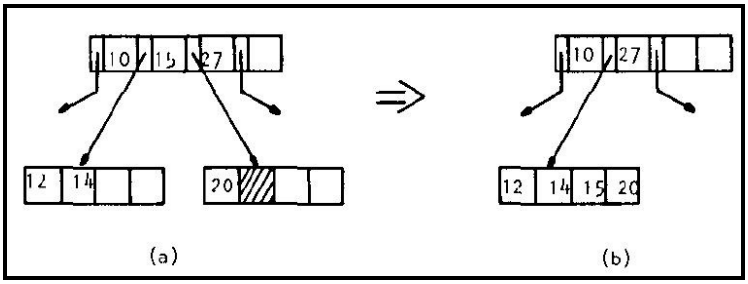

Figura 25. Uma remoção (a), causando concatenação e re-balanceando a árvore
Fonte: COMER, D. (1979)

Quando alguns nós perdem a chave separadora por causa da concatenação de duas de suas folhas, isto soma mais *underflow* e requer redistribuição de um de seus vizinhos. O processo de concatenação pode forçar a concatenação no próximo nível superior e então, na raiz. Finalmente, se os nós descendentes da raiz são concatenados, eles formam uma nova raiz, decrementando a altura raiz da B-Tree em 1 (um) nível (COMER, 1979).

4.1.3 B-Tree e Dados Espaciais

Dados espaciais são associados a coordenadas espaciais que se estendem a pontos, linhas, polígonos e objetos volumétricos. Dados espaciais são multidimensionais e conseqüentemente incluem dados dimensionais muito grandes. Porém, quando um campo é inicialmente investigado, o número de dimensões tende a diminuir e isto afeta o design dos índices para aplicações espaciais (OOI; TAN, 2002).

A representação de objetos e as estruturas de indexação são tratadas separadamente em aplicações de bancos de dados espaciais. Objetos de forma completa são aproximados por simples containeres para propósitos de recuperação. O objetivo de tal aproximação é filtrar falsas fontes quanto possível, antes de executar mais testes nos

complexos objetos espaciais. Estes objetos são indexados diretamente nos seus espaços naturais ou transformados e indexados em um espaço paramétrico.

Para a aproximação posterior, os dados espaciais são particionados em uma grade de células do mesmo tamanho, onde são numerados dentro de um padrão. A idéia é assimilar um número para cada grade em um espaço e esses números serem usados para se ter um número representativo para os objetos espaciais. As funções usadas nos mapeamentos devem preservar a proximidade entre os dados de uma forma ordenada o bastante para se ter uma boa busca espacial. Um objeto espacial com extensão é representado por um conjunto de números ou objetos uno-dimensional. Esses pontos uno-dimensional são indexados utilizando as B-Trees.

Um esquema proeminente é a *universal* B-Tree (UB-Tree). Nessa B-Tree, o esquema de ordenação Z (*Z-ordering*) é usado para mapear um espaço multidimensional para um espaço dimensional simples. O valor Z (*Z-Value*) de um ponto, computado pelo intercalamento dos bits de uma *string* de bits representa os valores das dimensões do ponto, é o número ordinal do ponto na curva Z (*Z-Curve*). Baseado no valor Z do ponto, ele pode ser armazenado em uma B^+-Tree.

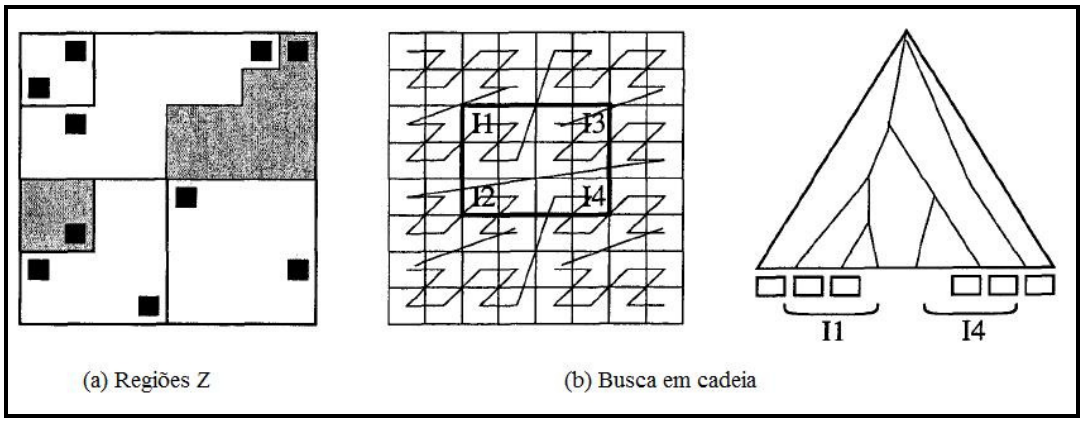

Figura 26. Regiões Z de uma B-Tree e sua busca em cadeia
Fonte: OOI, B. C.; TAN, K. (2002)

A novidade da UB-Tree está em como ela organiza os dados espaciais. Basicamente ela divide o espaço em regiões *Z* deslocadas, onde cada qual é um espaço

coberto por um intervalo na curva Z. Além disso, cada região Z corresponde exatamente a uma página no armazenamento secundário, como por exemplo, uma folha de uma B^+-Tree. A Figura 26 (a) mostra um conjunto de 10 pontos e 6 regiões Z assumindo que cada página pode segurar somente 2 pontos.

Para processar uma consulta multidimensional de escala, a B-Tree calcula as regiões Z que intersectam a consulta. Cada região Z corresponde para uma requisição na B^+-Tree armazenando os valores Z. A Figura 26 (b) mostra um exemplo. Enquanto é possível pré-calcular todos os intervalos de valores Z de uma consulta, pode ocorrer um excesso de armazenamento já que muitos deles podem entrar na mesma região *Z*. A UB-Tree evita o excesso de armazenamento construindo os intervalos dinamicamente durante o tempo de execução, processando as requisições página por página.

4.2 R-TREE

A R-Tree é uma estrutura de dados hierárquica derivada da B-Tree criada por Antonin Guttman em 1984. A diferença entre as duas estão na natureza das chaves: valores numéricos ou alfanuméricos simples, no caso das árvores-B e pontos extremos de retângulos, no caso das R-Trees (CÂMARA et al, 2005).

A estrutura de dados divide o espaço hierarquicamente alinhado e sobrepondo possivelmente, o MRE. Cada nó da R-Tree tem um número variável de entradas. Cada entrada em um nó não-folha armazena dois tipos de informações; um modo de identificar o nó-folha e o menor retângulo de todas as entradas no nó filho.

A R-Tree é similar a B-Tree quanto ao registro de índices nos seus nós folhas contendo ponteiros para os seus objetos de dados. A estrutura é designada para no caso de uma pesquisa espacial somente um pequeno conjunto de nós será consultado.

A R-Tree busca organizar o MRE. Este retângulo é formado a partir da observação dos limites geométricos mínimos e máximos do contorno do objeto e é expresso pelas coordenadas dos seus pontos inferior esquerdo e superior direito (Figura 27).

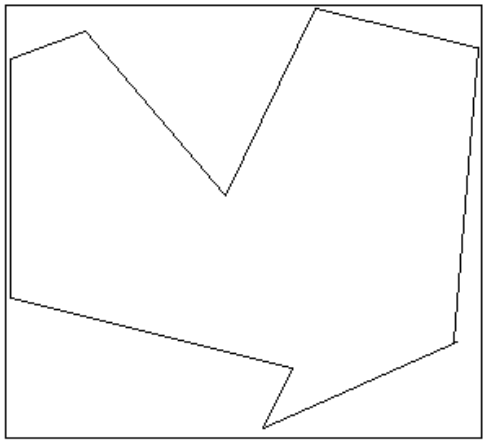

Figura 27. Objeto poligonal e seu MRE
Fonte: Adaptado de CÂMARA, G. et al (2005)

A R-Tree possui parâmetros que determinam à quantidade de chaves (retângulos) que poderão ocorrer em cada bloco de armazenamento, analogamente a B-Tree, em função do tamanho da página de armazenamento em disco. As regras básicas de formação de um R-Tree são muito semelhantes às B-Trees. Todas as folhas aparecem sempre no mesmo nível da árvore. Nós internos contem a delimitação de retângulos que englobam todos os retângulos dos nós nos níveis inferiores (Figura 28). Uma R-Tree de ordem (*m,M*) conterá entre *m* e *M* entradas em cada nó da árvore, exceto a raiz, que terá pelo menos dois nós.

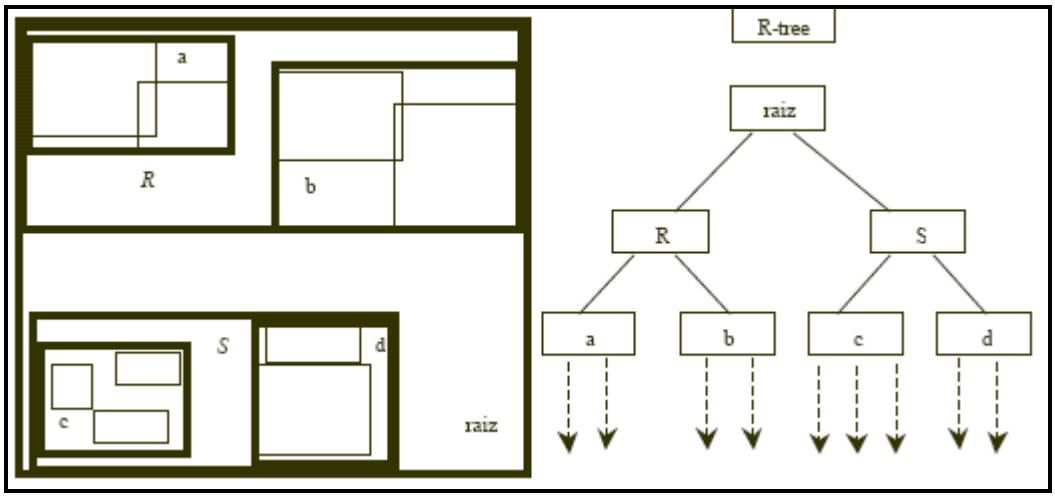

Figura 28. R-tree e seus retângulos correspondentes
Fonte: SILVA, R. (2002)

Os algoritmos de inserção e remoção utilizam o menor retângulo dos nós para assegurar que elementos "próximos" estejam no mesmo nó-folha. Cada entrada dentro do nó-folha armazena dois tipos de informações; um modo de identificar o elemento de dados atual e o menor retângulo do elemento de dados.

Semelhantemente, os algoritmos de busca usam o menor retângulo para decidir se procuram ou não dentro de um nó-filho. Deste modo, a maioria dos nós na árvore nunca são "tocados" durante uma busca. Como as B-Trees, isso faz da R-Tree satisfatória para banco de dados, onde os nós podem ser armazenados no disco quando necessário. Diferentes algoritmos podem ser usados para dividir os nós quando eles estão cheios, resultando nos sub-tipos de R-Tree chamados "quadrático" e "linear".

4.2.1 Propriedades da R-Tree

Um banco de dados espacial *n*-dimensional consiste de um amontoado de registros onde cada uma tem um identificar único, que o faz ser recuperado. Cada nó-folha contém um registro como *(I,* identificador-folha*)*. Neste caso o identificador-folha aponta para o lugar onde o objeto está armazenado no banco de dados espacial e *I* é um retângulo *n*-dimensional como $I = (I_0, I_1, ..., I_{n-1})$.

Todo I_k representa um intervalo de saltos fechado $[a_k, b_k]$ (Figura 29), os quais são os pontos iniciais e finais do retângulo no k-enésimo dimensão. Se um ou dois pontos finais do intervalo I_k são iguais ao infinito, isso significa que o objeto descrito estende externamente no *k*-enésimo dimensão. Então todo objeto espacial pode ser representado pelo MRE, o menor retângulo que contém o objeto designado.

Nós não-folha contém entradas *(I*, ponteiro-filho) onde o ponteiro-filho aponta para o nó-filho na R-Tree e *I* cobre todos os retângulos do nó-filho.

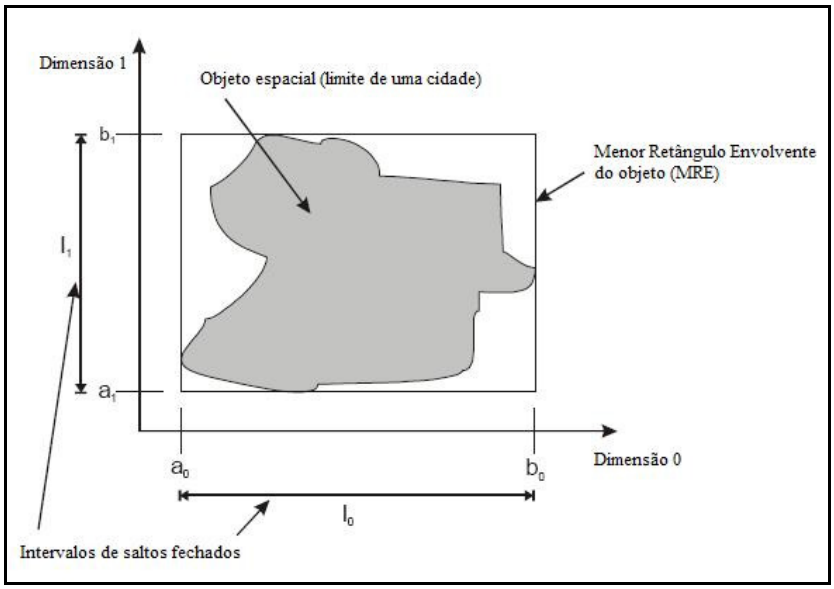

Figura 29. Intervalos de saltos fechados e o MRE
Fonte: Adaptado de KEMPER, A. (2001)

Se *M* (depende do tamanho da página do disco e das dimensões *n*) é o número máximo de entradas, isso se ajustará dentro de um nó e $m \leq M/2$ especificará o número mínimo de entradas em um nó, então uma R-Tree precisará seguir as seguintes propriedades:

a) todo nó-folha, que não seja a raiz, contém entre *m* e *M* registros de índices;

b) para cada registro de índice, *(I,* identificador-folha*)* em uma folha, *I* é o menor retângulo que contém o objeto *n*-dimensional representado neste registro;

c) cada nó não-folha, que não seja a raiz, contém entre *m* e *M* filhos;

d) para cada entrada *(i,* ponteiro-filho*)* em um nó não-folha, *I* é o menor retângulo que contém o nó filho;

e) o nó raiz tem pelo menos dois filhos, a menos que seja uma folha;

f) todas as folhas aparecem no mesmo nível.

A altura de uma *R-Tree*, com *N* registros de índices é $[log_m N]$-1, e no melhor caso cada nó contém no mínimo *m* nós-filho.

Um *overflow* pode ocorrer se o *m* é fixado muito alto (perto de *M*), de forma que o nó esteja densamente cheio. Se uma ou mais entradas serão escritas neste nó, o máximo número de entradas *M* será excedido e ocorrerá o *overflow* no nó, como visto na Figura 30 (a).

Equivalente ao *overflow* o *underflow* do nó ocorrerá quando *m* é definido muito largo e uma ou mais entradas são removidas de forma que o número de entradas irá remanescer sob *m*, como visto na Figura 30 (b).

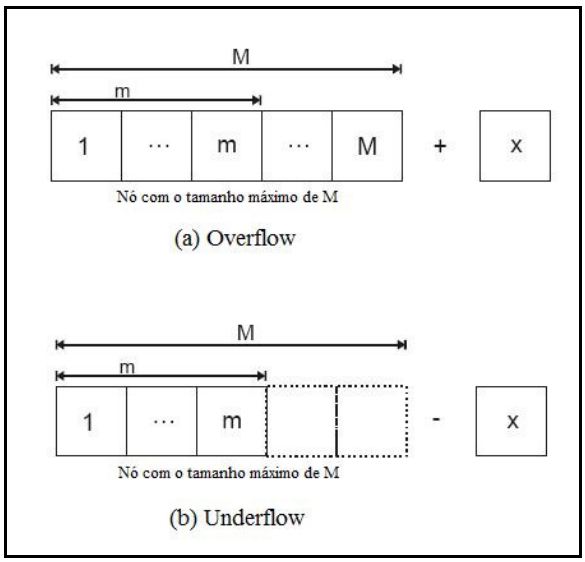

Figura 30. *Overflow* e *underflow* de um nó
Fonte: Adaptado de KEMPER, A. (2001)

As configurações de *m* e *M* são muito importantes para a eficiência do banco de dados. Enquanto *M* depende dos parâmetros que são determinados por meio do disco rígido de dados, propriedades nos quais os dados são armazenados. De outro lado, o tamanho de *m* é elementar para a performance do banco de dados. Se o banco de dados é necessário apenas para buscas com poucas atualizações, um *m* de tamanho maior será recomendado para manter a altura da R-Tree baixa e a performance de busca alta. Mas isso é porque o risco de um *overflow* ou *underflow* ocorrer é muito alto. No caso de *m* ser definido para um valor menor, o banco de dados será otimizado para atualizações freqüentes e modificações porque os nós não estão tão cheios como se o *m* fosse largo e o risco de ocorrer *overflow* ou *underflow* é obvio (KEMPER, 2001).

4.2.2 Operações Básicas

Guttman criou os algoritmos das operações elementares da R-Tree. Estes métodos são semelhantes aos seus equivalentes na B-Tree, somente o controle de *underflow* e *overflow* são diferentes dependendo da localização espacial dos dados (KEMPER, 2001).

Para explicar os algoritmos, será definido um banco de dados de dimensão 2. Os valores *m* e *M* são as entradas mínimas e máximas do nó (*m*=2, *M*=5). Neste banco de dados (Figura 31 a, b e c) *nome* é o nome do estudante, *semestre* é o semestre no qual o estudante está cursando e *créditos* é a soma dos créditos alcançados pelo aluno.

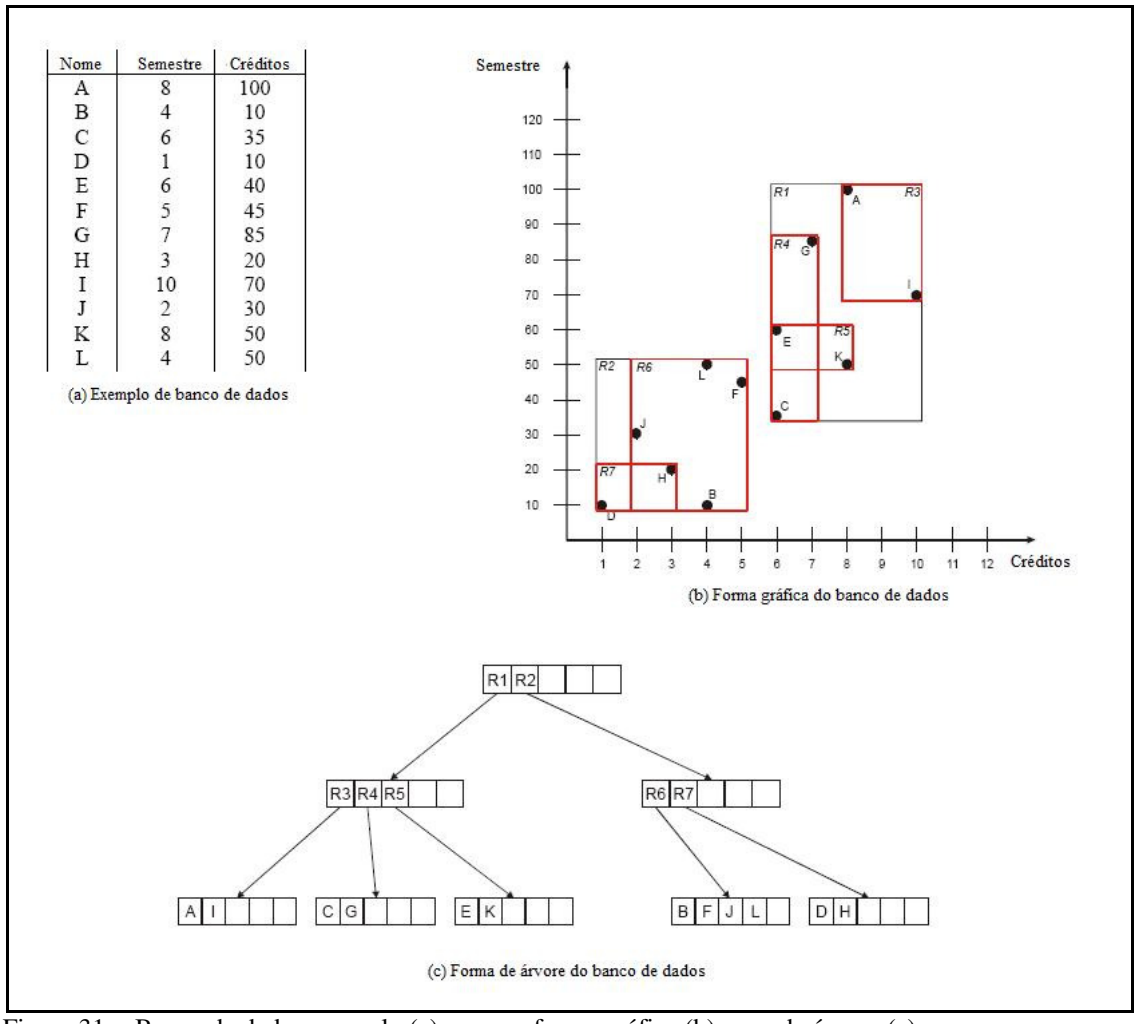

Figura 31. Banco de dados exemplo (a), em sua forma gráfica (b) e em de árvore (c)
Fonte: Adaptado de KEMPER, A. (2001)

4.2.2.1 Busca

A busca em uma R-Tree é similar a busca em uma B-Tree onde a busca se inicia na raiz. Mas para que isso aconteça em uma R-Tree, vários retângulos são sobrepostos, como exibido na Figura 32. Todos os seus retângulos serão visitados e não há uma garantia de boa performance nos piores casos. O algoritmo de busca funciona da seguinte forma:

a) se por exemplo exista uma árvore com a raiz *T* e deseja-se procurar um retângulo *S*:

b) se *T* não é uma folha, será aplicada a busca em todos os seus nós-filho de quem é apontado pelo ponteiro-filho e isso se sobrepõe com *S*;

c) se *T* é uma folha, retornará todas as entradas que sobrepõe com *S* como resultado.

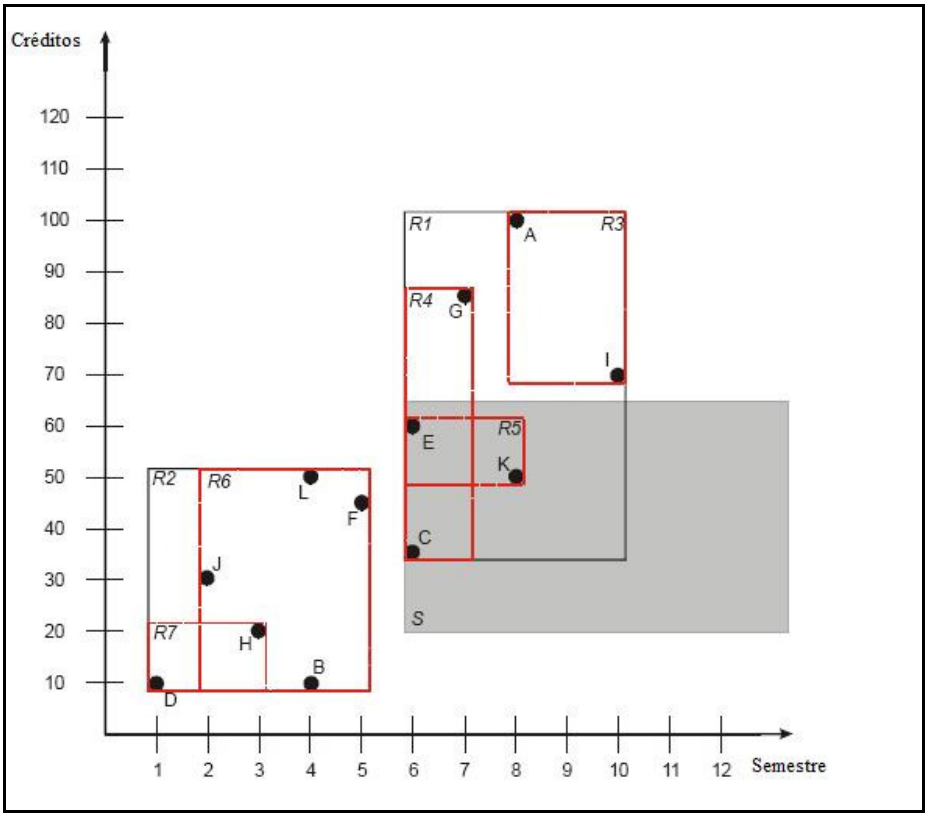

Figura 32. Busca do retângulo *S*
Fonte: Adaptado de KEMPER, A. (2001)

A Figura 32 acima exibe uma busca exemplo descrita por Kemper (2001), onde se deseja encontrar todos os estudantes que estão no 6º (sexto) semestre ou mais e que tenham entre 20 e 65 créditos.

R1 sobrepõe o retângulo *S*, não *R2*, então a busca se procede em *R1*. No próximo passo *R4* e *R5* são sobrepostos com *S* e nestes retângulos são encontrados o resultados. De *R4* é encontrado *C* e de *R5* é encontrado *E* e *K,* então o resultado da busca é (*C,E,K*).

Segundo Câmara et al (2005), pesquisas na R-Tree são relativamente simples de serem executadas. O único problema é que um grande número de nós pode ter que ser examinado, pois um retângulo pode estar contido em vários outros, mesmo que o objeto esteja contido em apenas um nó da folha.

4.2.2.2 Inserção

Se uma nova entrada será inserida no banco de dados, um novo registro de índice será adicionado a R-Tree. Este é o único modo da R-Tree crescer em altura, isto é, se há um estouro na capacidade de armazenamento do nó, este será divido. No caso da divisão atingir a raiz, a altura da árvore irá aumentar. O algoritmo de inserção é o seguinte:

a) *E* será a nova entrada;
b) o algoritmo de escolha de folha (*ChooseLeaf*) é usado para encontrar o nó folha *L* onde *E* será inserido;
c) se há espaço em *L, E* será inserido, senão é aplicado a divisão do nó (*SplitNode*) em *L*, que irá retornar *L* e *L'* contendo *E* e todos os elementos de *L;*
d) o algoritmo de ajuste da árvore (*AdjustTree*) é usado em *L* e se havia uma divisão antes o ajuste também é feito em *L'*;

e) se a divisão atingir a raiz e ainda ela tiver que continuar, será criado uma nova raiz de quem os filhos são dois nós que a divisão da raiz retornará.

f) O algoritmo de escolha de folha (*ChooseLeaf*) seleciona uma folha disponível para inserir a nova entrada *E*, deixando *N* ser o nó raiz. O algoritmo funciona da seguinte maneira:

g) Se *N* é uma folha, o *E* é devolvido;

h) se *N* não é uma folha, é procurada a entrada F_k em *N,* do qual o retângulo tem que ser aumentado para incluir o retângulo de *E*;

i) o algoritmo de busca de folha é aplicado em F_k até que uma folha seja alcançada.

O algoritmo de ajuste de árvore (*AdjustTree*) sobe de um nó folha *L* para a raiz, enquanto ajusta os retângulos e faz as divisões de nós necessárias. O valor de *N* é igual a *L.* Se *L* era anteriormente dividido *N'* é igual a *L'*. O algoritmo de ajuste funciona da seguinte forma:

a) Se *N* é a raiz, fim;

b) P será o parente de *N*. As entradas de *N'* são ajustadas em *P* de forma que se cubram todos os retângulos em *N*;

c) se uma divisão ocorrer, é adicionado uma nova entrada em *P* que apontará para *N'*. Se o nó parente *P* passar do limite de armazenamento, uma divisão de nó (*SplitNode*) é feita.

Um exemplo de inserção para esclarecer o funcionamento da inserção em R-Tree: os dados de um novo estudante (Q, 10, 65) serão adicionados ao banco de dados, o algoritmo de escolha de folha (*ChooseLeaf*) retornará *R1* como o primeiro nó novo. Logo após *R3* é escolhido como o retângulo onde será inserida a nova entrada e que não passará de seu limite

de armazenamento. Então *Q* é inserido em *R3*. Após isso o ajuste de árvore (*AdjustTree*) atualiza o retângulo *R3* e *R1*.

4.2.2.3 Remoção

Se um objeto será removido do banco de dados, é necessário encontrar o registro de índice correspondente a *E* e removê-lo. Este é o único modo de uma R-Tree diminuir a sua altura. O algoritmo de remoção (*Delete*) é o seguinte:

a) o algoritmo de busca de folha (*FindLeaf*) encontra a folha *L* que contém *E*. Fim se *E* não é encontrado;

b) *E* é removido de *L*;

c) o algoritmo de condensamento da árvore (*CondenseTree*) é usado em *L* para condensar abaixo dos nós cheios;

d) se a raiz tem apenas um único filho após o ajuste, o filho é nova raiz e a altura da árvore diminuirá.

O algoritmo de busca de folha (*FindLeaf*) encontra o nó folha que contém o registro de índice E, colocando T como o nó raiz. O algoritmo *FindLeaf* é descrito a seguir:

a) se *T* não é uma folha, a busca de folha é aplicada em todos os filhos cujos retângulos sobrepõem o de *E*. Se *E* é encontrado, ele será retornado;

b) se *T* é uma folha, todas as entradas são comparadas com *E* e se eles são iguais, *T* será retornado.

O algoritmo condensador de árvore (*CondenseTree*) pega um nó folha *L* da entrada que foi apagada e elimina o nó se ele tem poucas (menos que *m*) entradas. O algoritmo passa por cima da árvore inteira e ajusta todos os retângulos, se necessário, para

deixá-los menores. Este algoritmo é descrito a seguir, onde *N* é igual a *L* e *Q* é o conjunto dos nós eliminado:

a) Se *N* é a raiz, vá para o último passo (e), senão *P* será parente de *N*;

b) Se *N* tiver menos do que *m* entradas, as entradas *N* serão removidas em *P* e adicionadas em *Q*;

c) se não há *underflow* em *N*, o retângulo é ajustado para o *MRE* que contém todas as entradas remanescentes em *N*;

d) *N* é atribuído igual a *P*, retorno ao primeiro passo (a);

e) todas as folhas armazenadas em *Q* são inseridas na árvore usando a inserção (*Insert*). Todos os nós não-folhas que estão armazenados em *Q* precisam ser inseridos em um nível alto da árvore para que a árvore permaneça com a altura balanceada.

Por exemplo, se o estudante *K* fosse excluído do banco de dados. O algoritmo *FindLeaf* retornará *R5* e o seu retângulo correspondente, *K* será removido de *R5* e é avisado que *R5* ficará vazio. Após isso é aplicado o algoritmo *CondenseTree* em *R5*, que removerá *R5* da árvore, adicionando *E* a *R4* usando *Insert* e atualizando o retângulo *R1*.

Segundo Kemper (2001), a maioria dos procedimentos da R-Tree são similares aos equivalentes na B-Tree, mas em distinção, quando os nós são agrupados, os nós na R-Tree são reinseridos ao inverso. Isso é de comparável eficiência, e muito fácil para implementar pois o algoritmo existente *insert* pode ser usado. Na R-Tree, isso também é suportado ao apagar vários objetos em uma determinada área.

4.2.2.4 Divisão de Nós

Quando uma nova entrada é inserida em um nó cheio (contendo *M* entradas), é necessário dividir estes *M*+1 nós em dois novos nós. A meta da divisão será minimizar o tamanho dos dois retângulos resultantes porque a busca no retângulo menor terá mais chances de apenas visitar os nós necessários. Porque quanto menor o retângulo menor a possibilidade de ele sobrepor outros. Na figura 32 (a) vê-se um exemplo de uma boa divisão e na figura 32 (b) uma má divisão.

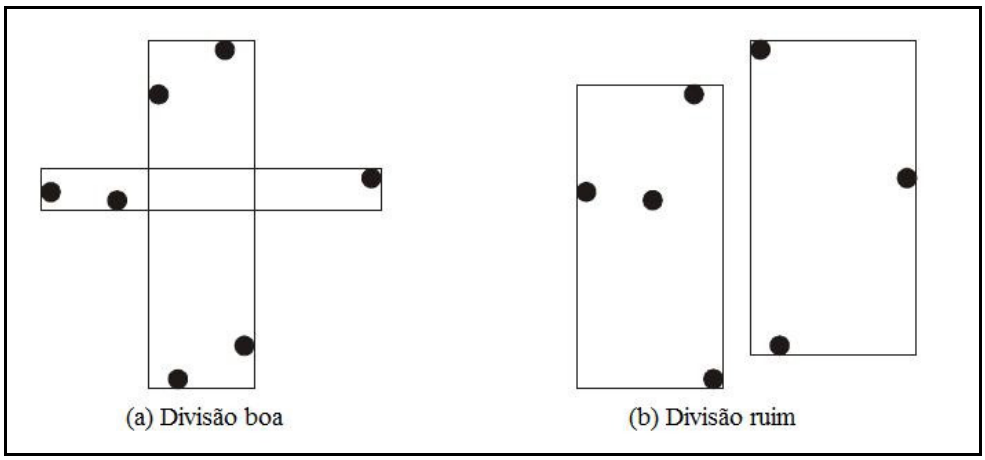

Figura 33. Uma divisão boa (a) e ruim (b)
Fonte: Adaptado de KEMPER, A. (2001)

Existem três algoritmos de divisão de nós, para dividir *M*+1 entradas em dois nós *N* e *N'*. Seus nomes dizem como o uso da CPU incrementa relativamente a *M* (GUTTMAN 1984 apud KEMPER, 2001).

O primeiro é o *Exhaustive SplitNode Algorithm* (algoritmo de divisão exaustiva) que passa diretamente por todas as possibilidades de agrupamento e escolhe o melhor. Ele tem a melhor qualidade, mas há 2^{M+1} possibilidades de combinações e o uso da CPU cresce exponencialmente. Segundo Kemper (2001), este método não é apropriado para grandes bases de dados onde *M* é muito grande.

O segundo algoritmo é o *Quadratic Cost Algorithm* (algoritmo de custo quadrático). Neste método as duas entradas que estão mais distantes da outra são colocadas em nós diferentes. Todos os elementos remanescentes são distribuídos considerando para o mesmo padrão. Para cada nó, o espaço requerido quando adicionado a *N* ou *N'* é calculado. Então o nó com a maior diferença entre este dois grupos será adicionado ao nó requerendo menos espaço. Neste algoritmo são usados dois métodos auxiliares: o *PickSeeds* que seleciona os dois primeiros elementos do grupo e o *PickNext* que seleciona uma das entradas restantes para colocar no grupo. Segundo Kemper (2001), este algoritmo não produz a melhor divisão, mas é o mais eficiente para bases de dados grandes.

O terceiro e último algoritmo chamado *Linear Cost Algorithm* (algoritmo de custo linear), escolhe para cada dimensão *i* pertencente a {0, ..., *k*-1} as duas entradas que estão mais distantes uma da outra nesta dimensão *i* e as colocam em dois nós diferentes. Após isto estar pronto para cada dimensão, os nós são distribuídos aleatoriamente. A diferença em relação ao algoritmo de custo quadrático está localizada nos métodos *PickSeeds* e *PickNext*. Segundo Kemper (2001), este algoritmo é muito eficiente, porém não é eficaz e apresenta uma busca ruim.

4.3 GIST

GiST significa *Generalized Search Tree* (Árvore Genérica de Busca) que é uma forma genérica de indexação. Ela é usada para agilizar a busca em todos os tipos de estrutura de dados irregulares (dados espaciais, arranjos de números inteiros, entre outros). É uma árvore balanceada que serve como modelo básico para implementar outros tipos de índices arbitrários. B-Trees, R-Trees e outros métodos de acesso podem ser implementados com GiST (HELLERSTEIN et al, 1995).

Cada nó da árvore contém um número de entradas $e = (p, ptr)$, onde p é o predicado que descreve a sub-árvore indicado por *ptr*. As sub-árvores recursivamente particionam os registros de dados. Porém, ela necessariamente não particiona os dados espaciais. GiST pode modelar então árvores de particionamento espacial ordenadas como as B-Trees e árvores de não-particionamento espacial desordenadas como as R-Trees (KEMPER, 1999).

4.3.1 Estrutura

Uma GiST é uma árvore balanceada com uma variável de carga entre kM e M, $^2/_M \leq k \leq {}^1/_2$, com exceção no nó raiz, que pode ter uma carga entre 2 e M. A constante k é o fator de enchimento mínimo da árvore. Nós folha contém (*p, ptr*) pares, onde p é o predicado que é usado como a chave de busca e *ptr* é o identificador de alguns registros do banco de dados. Nós não-folha contém (*p, ptr*) pares, onde p é o predicado como chave na busca e *ptr* aponta para outro nó da árvore. Predicados podem conter qualquer número de variáveis livres, tanto quanto cada registro referenciado pelas folhas da árvore pode instanciá-las.

4.3.2 Propriedades

As seguintes propriedades segundo Hellerstein et al (1995), são invariantes em GiST:

a) todo nó contém entre kM e M entradas de índice a menos que seja a raiz;

b) para cada entrada de índice (*p, ptr*) em um nó-folha, o p é verdadeiro quando instanciado com os valores do registro indicado;

c) para cada entrada de índice (*p, ptr*) em um nó não-folha, *p* é verdadeiro quando instanciado com os valores de qualquer registro alcançável de *ptr*;

d) a raiz tem pelo menos dois filhos, a menos que seja uma folha;

e) todas as folhas aparecem no mesmo nível.

4.3.3 Métodos fundamentais

GiST é uma árvore de estrutura balanceada como uma B-Tree, contendo (*p, ptr*) pares. Mas as chaves em GiST não são inteiros como em uma B-Tree. Ao invés disto, uma chave de GiST é um membro de uma classe definida pelo usuário e representa algumas propriedades de todos os itens de dados alcançáveis do ponteiro associado a chave. São necessários seis métodos para fazer uma GiST funcionar. Estes métodos auxiliam a árvore nas operações básicas de inserção, remoção e busca (HELLERSTEIN et al, 1995). Estes métodos são os seguintes:

a) **Consistent** (consistência) - *E,q*: dado uma entrada *E* = (*p, ptr*), e um predicado de análise *q*, retornará falso se *p^q* for considerado insatisfatório e retornará verdadeiro caso contrário;

b) **Union** (união) - *P*: dado um conjunto *P* de entradas (*p*1*, ptr*1), ... (*pn*, *ptrn*), retornará algum predicado *r* para todos os registros armazenados entre *ptr*1 e *ptrn*;

c) **Compression** (compressão) - *E*: dado uma entrada *E* = (*p, ptr*), retornará uma entrada (*pp, ptr*) onde *pp* é a representação comprimida de *p*;

d) **Descompression** (descompressão) - *E*: dado uma representação comprimida *E* = (*PP, ptr*), retorna uma entrada (*r, ptr*) onde *r* é a representação descomprimida de *pp*;

e) **Penalty** (penalidade) – *E*1, *e*2: dado duas entradas *E*1 = (*p, ptr*), e2 = (*p*2, *ptr*2), retorna uma penalidade específica para inserir *E2* na sub-árvore que tem como nó raiz *E*1, que é usado para ajudar no processo de divisão da operação de inserção;

f) **PickSplit** (escolha divisão) – *P*: dado um conjunto *P* de *M*+1 entradas (*p*, *ptr*), *P* será dividido em 2 conjuntos de entradas *P1* e *P2*, cada um com o tamanho *kM*, onde *k* é o fator de preenchimento mínimo.

4.3.4 Operações Básicas

Os métodos fundamentais apresentados na seção anterior devem ser providos pelo criador da classe fundamental. Os três métodos a serem apresentados nesta seção são providenciados pela GiST e podem invocar os métodos fundamentais requeridos. As chaves são comprimidas e descomprimidas quando são lidas em um nó.

4.3.4.1 Busca

Pode ser usada para procurar qualquer conjunto de dados com qualquer predicado atravessando toda a árvore tanto quanto necessário para encontrar o resultado. O algoritmo de busca é controlado pelo método *Consistent*, que retornará verdade se o predicado é satisfatório e falso no caso contrário. O mesmo método aplica-se para nós folha e não folha do índice. Este é a mais geral técnica de busca, semelhante a das R-Trees. Este algoritmo tem como saída todos os registros que satisfazem *q*. O algoritmo de busca GiST (*R, q*) é o seguinte:

a) *R* é a raiz de GiST, predicado é *q*;

b) se *R* não é uma folha, checar cada entrada *E* em *R* para determinar a consistência (*E,q*). Para todas as entradas que são consistentes, invocar busca na sub-árvore onde o nó raiz é referenciado por *E.ptr*;

c) se *R* é uma folha, checar cada entrada de *E* em *R* para determinar a consistência (*E,q*). Se *E* é consistente, é uma entrada qualificada. Neste ponto *E.ptr* pode ser buscado para checar *q*.

4.3.4.2 Inserção

A inserção garante que a árvore fique balanceada. Ela é similar as rotinas de inserção das R-Trees e B-Trees. O método *Penalty* é usada para escolher uma sub-árvore para inserir, o método *PickSplit* é usado para o algoritmo de divisão de nós e o método *Union* é usado para realizar mudanças com o objetivo manter as propriedades da árvore. O algoritmo de inserção é demonstrado a seguir onde o resultado é uma nova GiST resultante da inserção de *E* no nível l:

a) *R* é a raiz de GiST de entrada *E* = (*p, ptr*), nível *l*, onde *p* é o predicado tal como *p* que armazena todos os registros alcançáveis de *ptr*;

b) invocar o algoritmo *ChooseSubtree* para encontrar onde *E* deverá ir. *L* = *ChooseSubtree*(*R,E,l*);

c) se há lugar para *E* em *L*, inserir *E* em *L*. Senão invoque o algoritmo *Split*(*R,L,E*);

d) realiza as modificações necessários por meio do uso do algoritmo *AdjustKeys*(*R,L*).

ChooseSubtree (escolha de sub-árvore) pode ser usado para encontrar o melhor nó para inserção em qualquer nível da árvore. A saída do algoritmo será o nó no nível *l* mais adequado para armazenar a entrada com as características do predicado *E.p*. O algoritmo *ChooseSubtree* é demonstrado logo abaixo:

a) *R* é a raiz de GiSTentrada *E* = (*p, ptr*), nível *l*;

b) se *R* está no nível *l*, retornar *R*;

c) senão entre todas as entradas *F* = (*q, ptr'*) em *R* encontrar uma tal que *Penalty*(*F,E*) é mínima. Retornar *ChooseSubtree*(*F.ptr',E,l*).

O algoritmo de divisão utiliza o método *PickSplit* para escolher como dividir os elementos de um nó, incluindo o novo registro a ser inserido na árvore. Uma vez que os elementos são divididos em dois novos grupos, é gerado um novo nó para um dos grupos, o insere na árvore e atualiza as chaves sobre o novo nó. O resultado do algoritmo é uma GiST com *N* divisões em dois e *E* inserido. O algoritmo *Split* é mostrado a seguir:

a) *R* é a raiz de GiST com nó *N* e entrada *E* = (*p, ptr*);

b) invocar *PickSplit* na união dos elementos de *N* e {*E*}, colocar uma das duas partições no nó *N* e colocar a partição remanescente em um novo nó *N'*;

c) inserir entrada para *N'* no nó pai, $e_{n'}$ = (*q, ptr'*), onde *q* é a união de todas as entradas em *N'* e *ptr'* é um ponteiro para *N'*. Se há espaço para $E_{n'}$ no nó pai *N* então armazene $E_{n'}$. Caso contrário invoque *Split*(*R*, nó pai de $Ne_{n'}$)[2];

d) modificar a entrada *F* que aponta para *N*, de forma que *F.p* seja a união de todas as entradas em *N*.

O algoritmo *AdjustKeys* assegura as chaves sobre um conjunto de predicados armazenados para os registro abaixo e é propositalmente especifico. A saída deste algoritmo é uma GiST com os ancestrais de *N* contendo as chaves corretas e especificas: O algoritmo é explicado logo abaixo:

a) *R* é a raiz de GiST, *N* nó da árvore;

b) se *N* é a raiz, ou a entrada que aponta para *N* já tem uma precisa representação da união das entradas em *N,* então retorne;

c) caso contrário, modificar a entrada *E* que aponta para *N* de forma que *E.p* seja a união de todas as entradas em *N*. Então *AdjustKeys*(*R*, nó pai de *N*).

4.3.4.3 Remoção

O algoritmo de remoção mantém o balanço da árvore por meio do método *Union* e procura manter as chaves mais específicas o possível. Quando há uma ordem linear nas chaves, o algoritmo *CondenseTree* utiliza a técnica "peça emprestado ou se funda" das B^+-Trees. Casos contrários são utilizados as técnicas de reinserção das R-Trees. O algoritmo *CondenseTree* é o mesmo explicado na seção 4.2.2. O resultado do algoritmo é uma árvore com *E* removido, como mostrado a seguir:

a) *R* é a raiz de GiST com entrada *E* = (*p, ptr*);

b) procurar nó contendo a entrada, invocar busca(*R,E.p*) e encontrar o nó-folha contendo *E*. Parar se *E* não é encontrado;

c) remover *E* de *L*;

d) realizar mudanças, invocar *CondenseTree*(*R,L*);

e) se o nó-raiz tiver apenas um nó filho após a árvore ser ajustada, fazer do nó filho a nova raiz.

4.4 K-D TREE

A k-d tree é uma árvore binária, ou seja, cada nó interno possui dois descendentes não importando a dimensionalidade *k* do espaço envolvido. É uma alternativa a um problema mais genérico do que o geométrico dos bancos de dados espaciais: o de busca em vários atributos (CÂMARA, G. et al, 2005). No método k-d tree o espaço é dividido em setores cujas formas dependem da ordem em que são adicionados.

A Figura 34 mostra a criação de setores considerando as características do *k*-d tree, onde a partir de cada dado inserido é gerado uma divisão de setores. A partir do ponto inicial (Chicago) é feita uma divisão vertical a partir do eixo X, gerando o lado esquerdo e direito. Para que uma nova ramificação seja criada, deve-se considerar os eixos alternativamente, criando-se divisões de esquerda-direita, acima-embaixo. Não podem ser criadas duas divisões no lado esquerdo por exemplo.

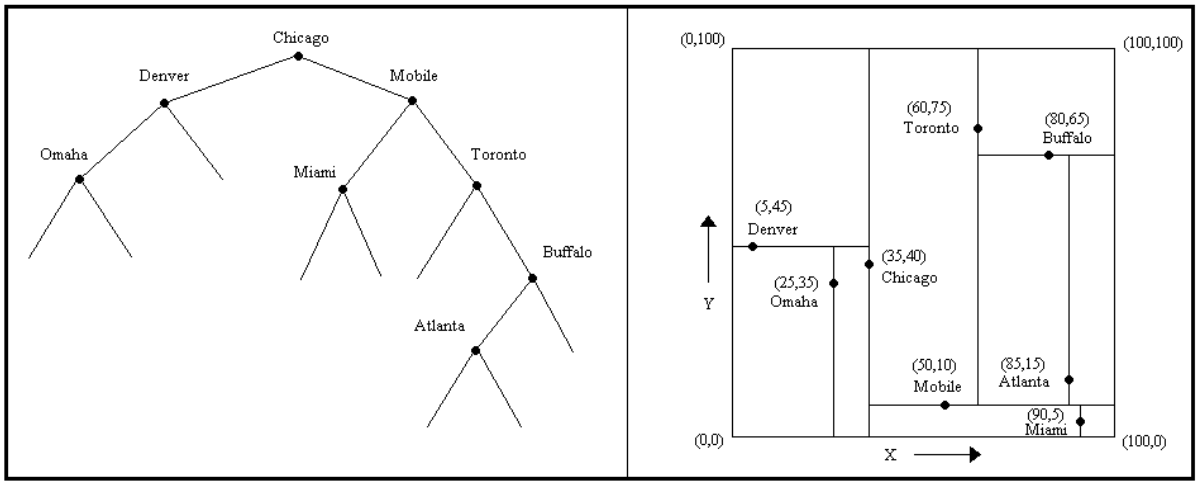

Figura 34. Representação de uma *k*-d tree
Fonte: Adaptado de SILVA, R. (2002)

Em uma chave formada por *k* atributos (*k* dimensões), onde k_i é o valor da i-ésima componente da chave, o conceito básico da *k*-d tree é a seguinte:

a) A cada nó da árvore é associado uma chave e um discriminador;

b) o discriminador é um valor entre 0 e k-1 que representa a dimensão ao longo do qual o espaço será subdividido, isto é, o componente k_i; que será empregado para definição da ordem da árvore;

c) todos os nós do mesmo nível são associados ao mesmo valor de discriminador;

d) o nó raiz está associado as discriminador 0, aos dois descendentes diretos, discriminador 1e assim por diante, até o k-ésimo que está associado ao discriminador k-1;

e) após o nível k+1 o ciclo começa a ser repetido, sendo associado o discriminador 0 a este nível;

f) a ordem definida por esta árvore é tal que, dado um nó qualquer P de discriminador j, qualquer nó descendente (L) da sub-árvore esquerda possui valor de chave k_j menor do que o k_j do nó P;

g) da mesma forma, qualquer nó descendente (R) da sub-árvore direita possui valor da chave k_j maior do que o k_j do nó P.

4.5 QUAD-TREE

A quad-tree (Figura 35) é um tipo de estrutura de dados organizada em árvore, em que cada “nó” ou “tronco” gera sempre quatro “folhas”. O SIG considera que cada nó corresponde a uma região quadrada do espaço. Esta região será subdividida (Figura 36) novamente em quatro partes gerando mais um nível na árvore e assim constantemente até que se chegue a ter um ou nenhum objeto dentro dos quadrados resultantes da subdivisão.

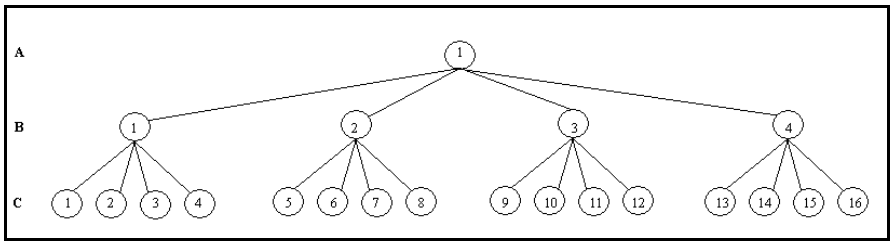

Figura 35. Quad-tree
Fonte: Adaptado de CÂMARA, G. et al (2005)

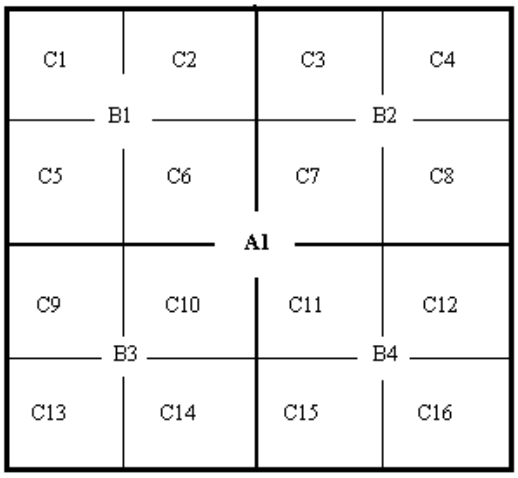

Figura 36. Subdivisão do espaço por uma quad-tree
Fonte: Adaptado de CÂMARA, G. et al (2005)

O processamento de consultas por região nessa estrutura inicia-se a partir da raiz, selecionando apenas as ramificações cujos quadrados estiverem em contato com o retângulo da região. Se nenhuma estiver, todas as ramificações abaixo daquele nó são descartadas. Se alguma estiver, toda a ramificação abaixo daquele ponto é selecionada e sucessivamente testada, até que se alcancem os objetos espaciais.

4.6 GRID FILES

Segundo Câmara et al (2005), o *hashing* é uma das técnicas mais interessantes para facilitar a pesquisa em dados convencionais. Chamado também de transformação de chave, consiste em criar uma série de receptáculos (*buckets*), que receberão os identificadores dos itens que se quer pesquisar. Estes buckets são numerados de forma seqüencial de 1 a n

(Figura 37) sob a forma de arranjo. Cada identificador que chega é transformado em um número de 1 a *n*, identificando o *bucket* correspondente a ele. No caso de uma inserção no *bucket*, Quanto maior for o valor de *n*, mais itens existirão dentro de cada *bucket* e o resultado da pesquisa será mais rápido.

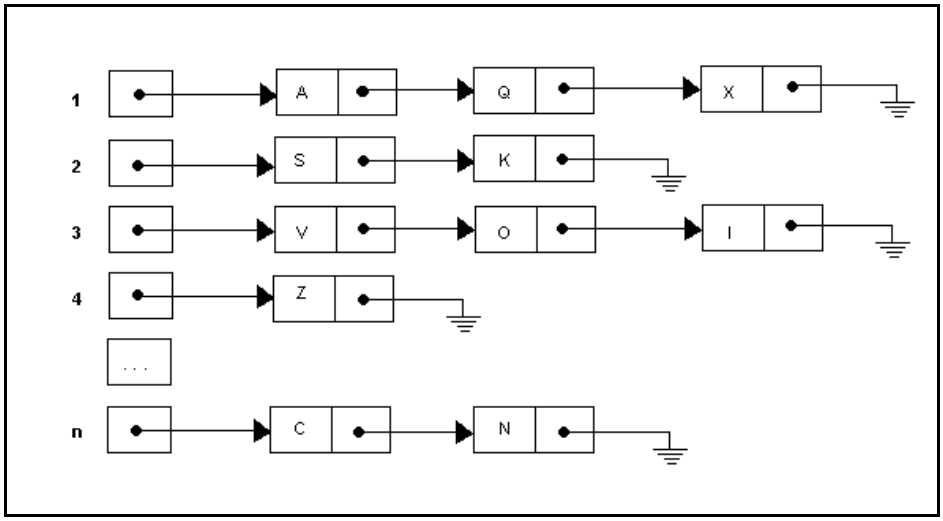

Figura 37. Hashing
Fonte: Adaptado de CÂMARA, G. et al (2005)

O método de acesso chamado *Grid File* (Figura 38) amplia o conceito de *hashing* para duas dimensões. Ao invés de se usar n buckets, uma grade regular irá cobrir toda a área de pesquisa. Ele possibilita pesquisas convencionais, porém utiliza duas variáveis distintas. A partir de cada elemento da grade são organizadas listas lineares contendo os identificadores dos objetos que estão localizados naquela área.

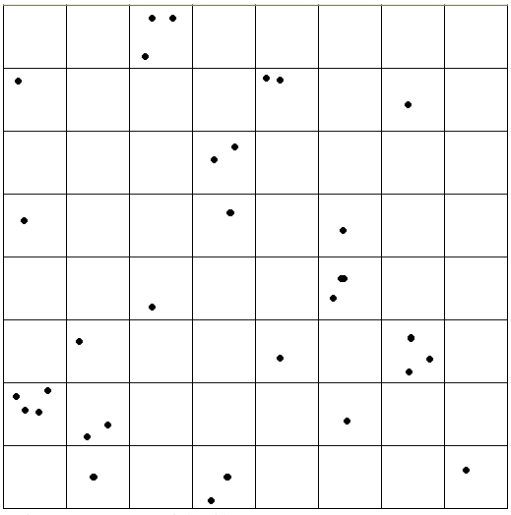

Figura 38. Grid File
Fonte: Adaptado de CÂMARA, G. et al (2005)

5 TRABALHOS CORRELATOS

Cifferi e Salgado publicaram em 2001 um artigo chamado “Análise da Eficiência de Métodos de Acesso Espaciais em Termos da Distribuição Espacial dos Dados”. Neste trabalho eles investigam o desempenho de um conjunto de MAE em relação a consultas de seleção (*point queries* e *range queries*).

O conjunto de métodos analisados neste artigo consiste em R-Tree, R-Tree Greene, R^+-Tree, Hilbert R-Tree, SR-Tree e três variantes da R*-Tree denominadas respectivamente de R*-Tree CR, R*-Tree FR e R*-Tree WR. Os dados espaciais utilizados para os testes foram compostos exclusivamente de retângulos.

Segundo Ciferri e Salgado (2001) o método é considerado competitivo se não é recomendável a sua troca pelo MAE que obteve o melhor resultado de desempenho, ou seja, quando a perda de desempenho for no máximo de 24,81%. Se a perda de desempenho for maior que 33,33% o método é considerado não-competitivo. Um método com perda neste intervalo é considerado pouco-competitivo.

O resultado do artigo indica o MAE R*-Tree como o mais eficiente de todos, sendo que a R-Tree e a R^+-Tree também apresentaram resultados positivos. Os métodos que obtiveram os piores resultados foram respectivamente a SR-Tree e Hilbert-Tree.

6 AMBIENTE DE TESTES

Para tornar possível o principal objetivo do projeto, um aplicativo e um banco de dados geográficos foram desenvolvidos para a análise da eficiência dos métodos de acesso espaciais. Após a realização desta etapa, foi possível executar os testes e analisar os resultados obtidos pela aplicação desenvolvida. As seções seguintes descrevem as ferramentas utilizadas no desenvolvimento e nos testes.

6.1 JAVA

O Java foi criado a partir de um projeto voltado para dispositivos inteligentes da SUN Microsystems. Ele foi lançado em 1995 como uma opção para construção de aplicativos para a *web* e acabou tornando-se um sucesso acadêmico. Sua plataforma possui uma linguagem com mesmo nome e é totalmente orientado a objetos com um enorme conjunto de classes. Java ainda é independente de plataforma, de ótimo desempenho e seguro. Atualmente ela possui três divisões: Java 2 *Micro Edition* (J2ME) para aplicações de dispositivos móveis, Java 2 *Standard Edition* (J2SE) para criação de aplicações de pequena e médio porte e Java 2 *Enterprise Edition* (J2EE) para o desenvolvimento de aplicações organizacionais de grande porte (JUNDL, 2006).

Segundo Jundl (2006) sua Interface de Programação de Aplicativos (API) já foi modificada várias vezes, porém a linguagem em si continua sem muitas mudanças. A última versão (5.0) incorpora vários elementos da nova linguagem (Tabela 1).

Tabela 1. Elementos incorporados ao Java a cada versão

Ano	Versão	Versão Interna	Novas características
1996	1.0	1.0	Versão inicial
1997	1.1	1.1	Classes internas
1998	1.2	1.2	Declaração strictfp, compilação JIT
2000	1.3	1.3	Tecnologia HotSpot para Java Virtual Machine
2002	1.4	1.4	Diretiva assert
2004	5.0	1.5	Genéricos, autoboxing, metada (anotações), enumerações

Fonte: JUNDL, P. J. (2006)

Java é independente de plataforma, pois seus aplicativos são compilados em um formato neutro chamado *bytecodes* e armazenados em arquivos de classes (.class). Para cada Sistema Operacional deve haver uma *Java Virtual Machine* (JVM) que interpreta os *bytecodes* como códigos nativos do Sistema Operacional. A Figura 39 ilustra isso.

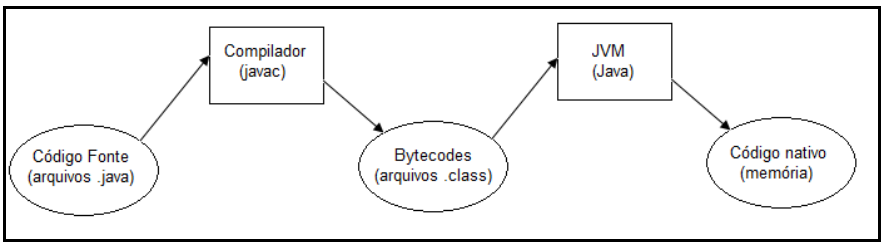

Figura 39. Código-fonte, *bytecodes* e código nativo
Fonte: Adaptado de JUNIOR, P. J. (2006)

O ambiente Java (Figura 40) é constituído de uma JVM, da API Java e das classes da aplicação. O Sun Java 2 *Runtime Environment* (J2RE) provê a JVM mínima para que seja possível executar as aplicações Java. Para desenvolvimento o ambiente utilizado é o J2SE *Development Kit* (JDK) que inclui um conjunto de ferramentas para desenvolvimento além do próprio J2RE.

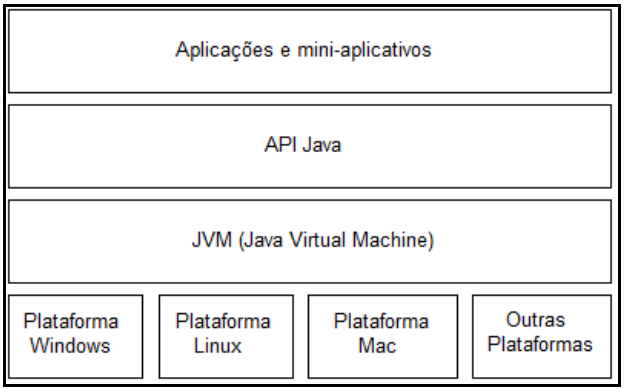

Figura 40. Ambiente Java
Fonte: JUNIOR, P. J. (2006)

6.1.1 JDBC

O Java Database Connectivity (JDBC) é uma camada de abstração que permite a uma aplicação Java utilizar uma interface padrão que adere ao ANSI-2 SQL para se conectar a uma base de dados por meio da linguagem SQL (RAMON, 2000).

Uma aplicação Java utiliza uma única API JDBC (Figura 41) que é independente em relação do *driver* ou ao banco de dados utilizados. Os desenvolvedores de aplicações apenas devem saber como utilizar a API JDBC e de que forma o *driver* adequado se conecta ao banco de dados. Estes *driver*s para acesso são fornecidos pelos seus fabricantes e/ou por terceiros.

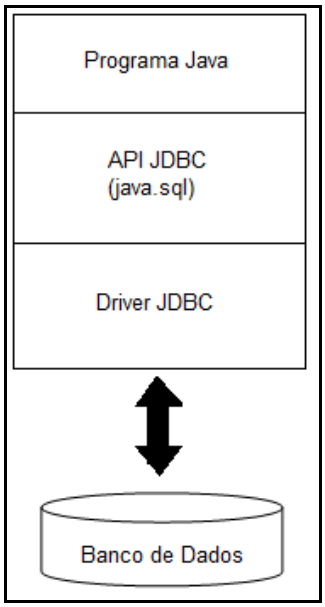

Figura 41. Arquitetura JDBC
Fonte: RAMON, F. (2000)

Para fazer a aplicação Java acessar o SGDB, é necessário um *driver* JDBC que implemente a interface padrão da API JDBC. Segundo Ramon (2000), os *driver*s JDBC são programas que fazem a intermediação entre um programa Java e um SGDB. Estes *driver*s são gerenciador pelo *DriverManager* que é responsável por gerenciar o conjunto de *driver*s JDBC disponíveis para o aplicativo Java, realizar a conexão do *driver* com o banco de dados e controlar o *login* e as mensagens entre os dois. O programa Java pode utilizar

simultaneamente vários *driver*s JDBC distintos e também acessando bancos de dados diferentes. Quando o aplicativo solicita uma conexão, o *DriverManager* determina qual o *driver* responsável por aquela conexão.

Os *driver*s JDBC são classificados em quatro tipos: ponte JDBC-ODBC com *driver* ODBC, *driver* com API parcialmente nativa, *driver* puro-Java JDBC-rede e *driver* puro-Java com protocolo nativo.

6.1.1.1 Ponte JDBC-ODBC com *driver* ODBC

Assim como O JDBC, a Open Database Connectivity (ODBC) é uma interface de acesso a banco de dados que possui *driver*s para diversos SGBDs. Um aplicativo Java pode acessar diretamente os *driver*s ODBC, porém isso implicará em executar chamadas a rotinas na linguagem de programação C da API do ODBC dentro da aplicação. Segundo Ramon (2000), chamadas com código nativo em Java são trabalhosas e eliminam a portabilidade de um programa.

A ponte possui código nativo a fim de fazer chamadas a API ODBC. Tanto a ponte quanto o *driver* ODBC devem residir na máquina onde o aplicativo Java estiver instalado (Figura 42), uma vez que as chamadas são sempre locais. Isto exige um maior controle sobre a distribuição do programa (RAMON, 2000).

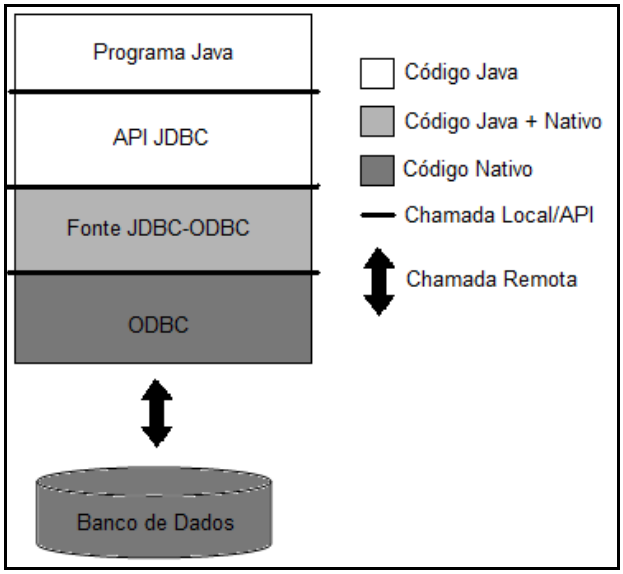

Figura 42. Ponte JDBC-ODBC com *driver* ODBC
Fonte: RAMON, F. (2000)

6.1.1.2 *Driver* com API parcialmente nativa

Tipo de acesso semelhante a ponte JDBC-ODBC. O *driver* JDBC acessa o aplicativo cliente do SGDB (Figura 43). Este acesso é geralmente feito com código nativo. Em cada estação deve ter instalado o aplicativo cliente. Isso exige um controle maior sobre a distribuição do aplicativo Java e como o *driver* depende do programa cliente, o procedimento para a sua instalação é informada pelo fornecedor (Ramon, 2000).

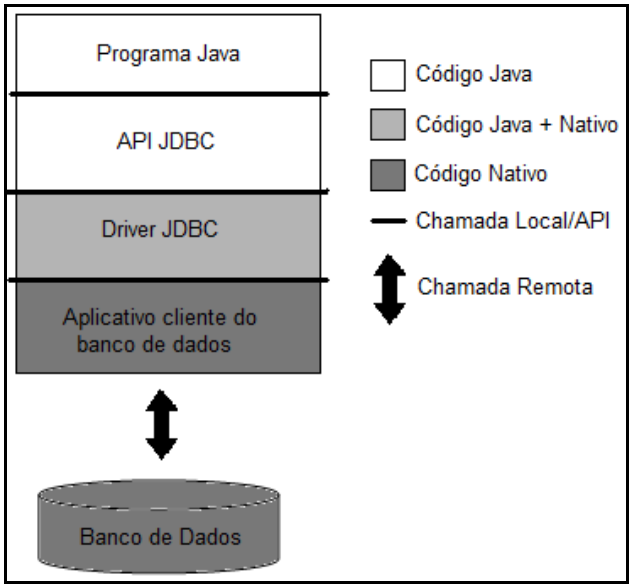

Figura 43. *Driver* com API parcialmente nativa
Fonte: RAMON, F. (2000)

6.1.1.3 *Driver* puro-Java JDBC-Rede

Segundo Ramon (2000) este é o tipo mais flexível de *driver*, pois pode ser totalmente escrito em Java. O *driver* acessa um servidor usando um protocolo neutro. O servidor estabelece a conexão com um ou mais aplicativos (Figura 44). Este *driver* pode acessar bancos de dados protegidos por *firewall* pelo fato de seu protocolo ser neutro e em geral, não é necessário instalar o *driver* já que é uma classe Java que pode ser carregada como as outras.

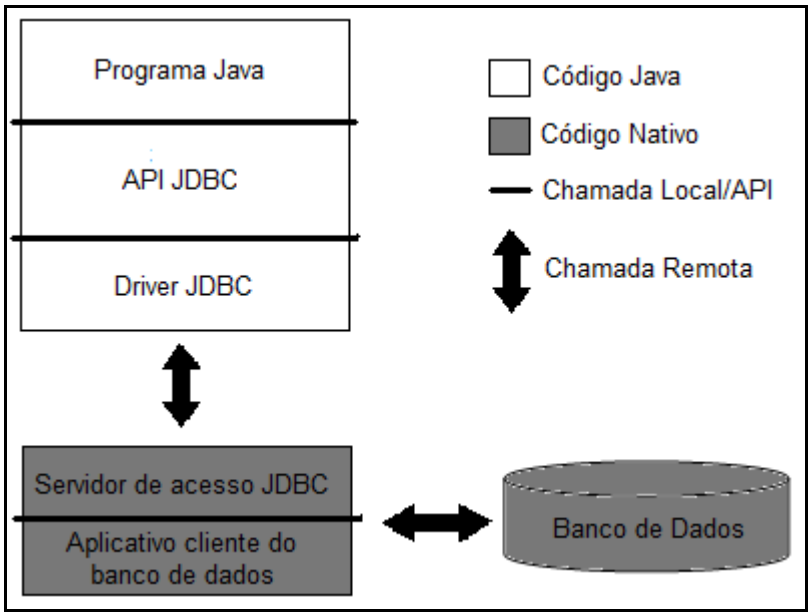

Figura 44. *Driver* puro-Java JDBC-Rede
Fonte: Adaptado de RAMON, F. (2000)

6.1.1.4 *Driver* puro-Java com protocolo nativo

Este acesso pode ser totalmente escrito em Java. O *driver* acessa diretamente o SGDB utilizando seu próprio código nativo (Figura 45), e por esta razão estes *driver*s são geralmente disponibilizados pelos fabricantes de bancos de dados. Ele é carregado no aplicativo como se fosse uma classe Java qualquer e não é necessária nenhuma instalação.

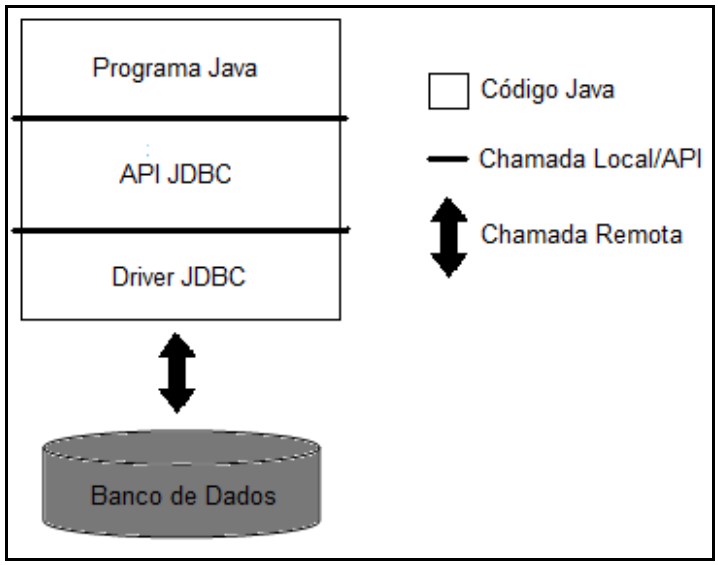

Figura 45. *Driver* puro-Java com protocolo nativo
Fonte: Adaptado de RAMON, F. (2000)

6.1.2 NetBeans IDE

NetBeans é uma plataforma para o desenvolvimento de aplicações Java para *desktop* e um ambiente de desenvolvimento integrado (IDE). A Plataforma NetBeans permite desenvolver aplicações a partir de um conjunto de componentes de software chamados módulos. Um módulo é um arquivo Java que contém a classe Java escrita para interagir com os APIs Abertos do NetBeans um arquivo de manifesto que identificam isto como um módulo.

O NetBeans IDE é compatível com muitas plataformas incluindo o Windows, Linux, Mac OS e Solaris. Dentre suas principais características podem ser destacadas as seguintes:

a) Construtor de interface gráfica do utilizador (GUI) Swing: desenvolvimento intuitivo utilizando Swing, “arrastando e soltando” os componentes da paleta Swing para a tela;

b) Desenvolvimento de aplicativos da *web*: facilidade de criação de páginas JavaServer Faces (JSF) e manipulação de bancos de dados e monitor para

Protocolo de Transferência de Hipertexto (HTTP) para depuração dos aplicativos *web*;

c) Controle de versão: o IDE reconhece automaticamente os diretórios de trabalho do Sistema de Versões Concorrentes (CVS);

d) Desenvolvimento colaborativo: projetos podem ser compartilhados em tempo real pela rede;

e) Editor avançado de códigos-fonte: o editor da linguagem recua, completa e faz realce sintático do código-fonte. Analisa o código, combina palavras e colchetes, marca erros e exibe dicas e o *javadoc*;

f) Gerador de perfis: oferece auxílio para otimizar o desempenho da aplicação

g) Suporte a linguagem de programação C/C++: amplia toda a funcionalidade para desenvolvimento de aplicativos na linguagem de programação C/C++.

6.2 POSTGRESQL

O SGDB PostgreSQL, juntamente com a extensão PostGIS, foram utilizados para o armazenamento dos bancos de dados geográficos. Este SGBD foi escolhido, pois dentre todos os SGDBs gratuitos (softwares livres), é o único que possui uma extensão robusta (também gratuita) para o tratamento de dados espaciais.

6.2.1 Características

O PostgreSQL é um sistema de banco de dados objeto-relacional livre e seu código-fonte é aberto. Ele contém as características de sistemas bancos de dados comerciais, porém com características melhoradas. Ele é compatível com a maioria dos sistemas

operacionais incluindo Linux, Windows e UNIX (AIX, BSD, HP-UX, SGI IRIX, Mac OS X, Solaris, Tru64). Sua implementação de SQL está de acordo com o padrão ANSI-SQL 92/99. Possui interfaces de programação nativas para as linguagens de programação C/C++, Java, .NET, Perl, Python, Ruby, Tcl, ODBC, entre outros e uma documentação claramente especificada.

PostgreSQL contém sofisticadas características como Controle de Concorrência de Multi-Versão (MVCC), *tablespaces*, replicação assíncrona, *backups online*, um sofisticado planejador/otimizador de consultas, suporte a caracteres internacionais e é altamente escalável quanto a quantidade de dados e ao número de usuários conectados. Alguns dos limites gerais do PostgreSQL estão listados na Tabela 2.

Tabela 2. Limites gerais do PostgreSQL

LIMITE	VALOR
Tamanho máximo do banco de dados	Sem limite
Tamanho máximo de uma tabela	32 TeraBytes
Tamanho máximo de uma linha	1.6 TeraBytes
Tamanho máximo de um campo	1 GigaByte
Número máximo de linhas por tabela	Sem limite
Número máximo de coluna por tabela	250 - 1600 (Depende to tipo da coluna)
Número máximo de índices por tabela	Sem limite

Fonte: POSTGRESQL, 2007.

Existe também um conjunto de extensões e características avançadas. As colunas são auto-incrementadas por sucessão e um limitador/compensador proporciona o retorno de um conjunto parcial de um resultado. O PostgreSQL suporta índices combinados, únicos, parciais e funcionais que utilizam os métodos de armazenamento B-Tree, R-Tree, Hash e GiST.

Segundo Câmara et al (2005), uma importante característica do PostgreSQL é o seu mecanismo de extensibilidade que permite incorporar capacidades adicionais ao sistema para torná-lo mais flexível para cada caso de aplicação.

6.2.2 Arquitetura

Quando uma sessão do PostgreSQL é iniciada, três processos trabalham de forma cooperativa (NIEDERAUR, 2004):

a) um processo *daemon* (*postmaster*);

b) a aplicação do cliente (como por exemplo o PSQL);

c) um ou mais servidores de banco de dados (o próprio processo postgres).

Um único processo *postmaster* gerencia os bancos de dados existentes em um computador. Os aplicativos que pretendem acessar o banco de dados devem fazer chamadas a biblioteca chamada LIBPQ. Esta biblioteca envia a requisição do usuário pela rede para o processo *postmaster*, que cria um novo processo-servidor e conecta o processo-cliente ao servidor. A partir desta operação os processos-cliente e servidor (respectivamente chamados de *frontend* e *backend*) se comunicam sem a intervenção do *postmaster*. Como *postmaster* é o gerenciador de conexões do PostgreSQL, ele é um processo que está sempre em execução. A Figura 46 exibe como ocorre a conexão.

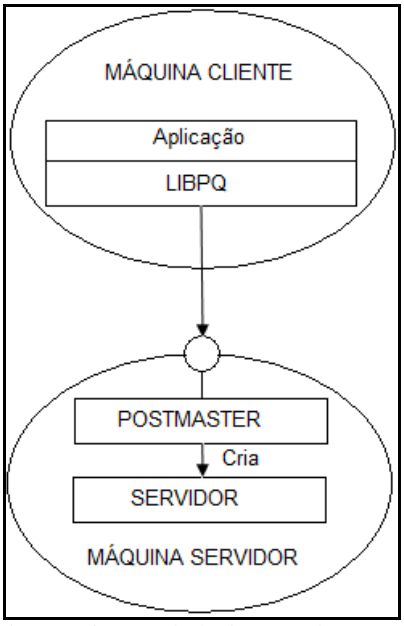

Figura 46. Estabelecimento de uma conexão no PostgreSQL
Fonte: NIEDERAUER, J. (2004)

6.2.3 Dados Espaciais

O PostgreSQL em sua versão oficial, apresenta suporte a dados geométricos (Figura 47), operadores espaciais simples e indexação espacial por meio do uso de uma R-Tree nativa ou implementada no topo do método GiST (HELLERSTEIN et al 1995 apud CÂMARA et al, 2005).

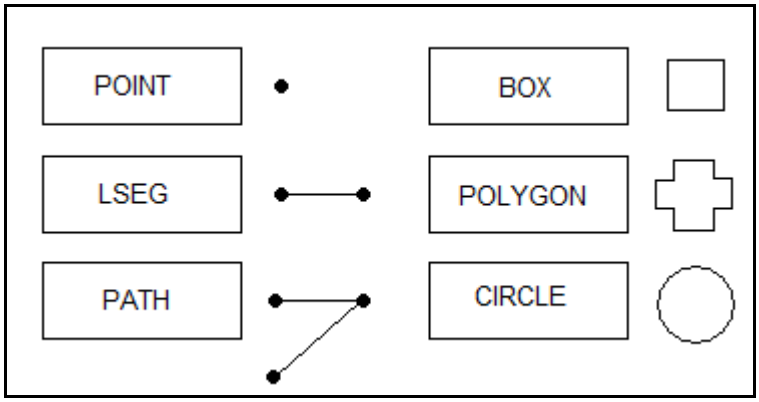

Figura 47. Tipos geométricos do PostgreSQL
Fonte: CÂMARA, G. et al (2005)

A R-Tree nativa do PostgreSQL possui uma notável limitação no seu uso, uma coluna to tipo *polygon* não pode exceder 8 KiloBytes (KB). Segundo Câmara et al (2005), é muito comum na prática em um SIG os polígonos ultrapassarem este tamanho, o que torna o seu uso quase inviável. Uma alternativa adotada na implementação do PostgreSQL é o uso da R-Tree implementada sobre o método GiST, já que este não possui limitações no tamanho do índice a ser indexado.

Existem poucos operadores e representações espaciais no PostgreSQL e a implementação de um SIG por meio destes requer muito esforço e a implementação de novos operadores e representações para a construção do sistema. Mas como explicado na seção 6.2.1, um dos pontos fortes do PostgreSQL é a sua extensibilidade e, segundo Câmara et al (2005), isso possibilitou o desenvolvimento de uma extensão geográfica mais completa, chamada PostGIS.

6.2.4 PostGIS

PostGIS é uma extensão espacial de código aberto para o SGDB PostgreSQL. Ele permite o armazenamento de objetos espaciais (SIG) no banco de dados que segue as especificações Simple Features Specification for SQL (SFSSQL) da Open Geoscience Consortium (OGC). Atualmente o seu desenvolvimento é mantido pela Refractions Inc, uma empresa que realiza consultoria em SIG e bancos de dados localizada no Canadá. Sua primeira versão foi lançada em 2001 pela Refractions sobre a licença GNU (REFRACTIONS, 2007).

6.2.4.1 Características

A implementação do PostGIS é baseada em geometrias de "peso-leve" e índices otimizados para reduzir o espaço de armazenamento e o trabalho computacional. Algumas das principais características inclusas do PostGIS estão listadas abaixo:

a) tipos geométricos (Figura 48) para pontos, *linestrings*, polígonos, multipontos, *multilinestrings*, multipolígonos e coleções geométricas;
b) predicados espaciais para determinar as interações entre os objetos geométricos;
c) operadores espaciais para mensurar objetos espaciais como área, distância, comprimento e perímetro;
d) operadores espaciais para determinar relações geométricas como união, diferença, diferença simétrica e *buffers*;

e) método de indexação GiST sobre R-Tree para realizar consultas com maior eficiência;

f) suporte para indexação seletiva, para prover maior desempenho nas consultas que mixam dados espaciais com não-espaciais.

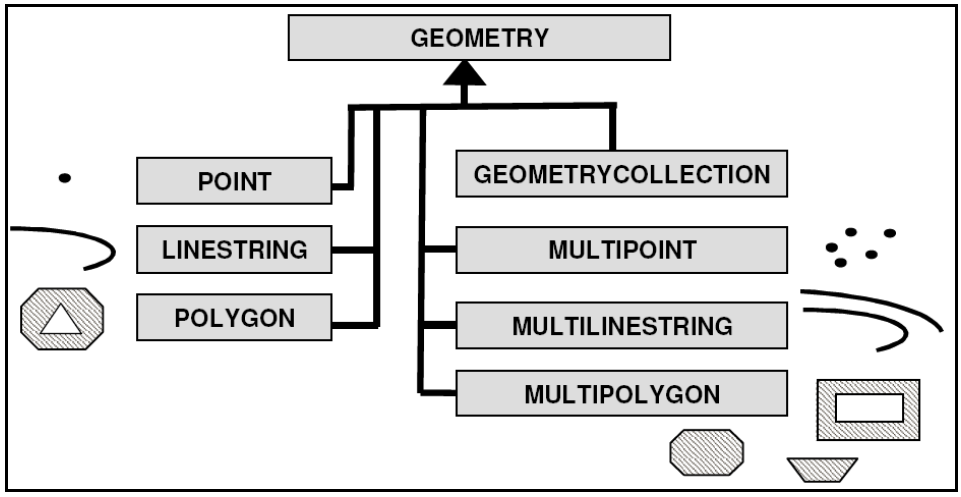

Figura 48. Tipos de dados espaciais do PostGIS.
Fonte: CÂMARA, G. et al (2005).

6.2.4.2 Suporte as características OGC

O suporte do PostGIS a objetos são características simples definidas pelo OGC. O PostGIS atualmente suporta as características e a representação de APIs, mas não as várias comparações e operadores de convolução cedidos na especificação SFSSQL da OGC. A especificação OpenGIS define dois caminhos padrões de objeto espacial de expressão: a forma Well-Know Text (WKT) e a forma Well-Known Binary (WKB). Estes incluem informações sobre o tipo do objeto e a coordenada que é formada (Refractions, 2007). Logo abaixo segue exemplos de representação textual das coordenadas dos objetos, onde (n1 n2) representam respectivamente (x y):

a) POINT(0 0)

b) POLYGON((0 0, 1 4, 4 1, 1 0),(1 2, 1 5, 5 5, 1 2))

c) LINESTRING(1 1, 2 2)

A especificação OpenGIS (padrão do OGC), define que o formato de armazenamento interno deve incluir um identificador sistema de referência espacial, o SRID. Ele é requerido ao criar objetos espaciais para a inserção no banco de dados.

6.2.4.3 SFSSQL

O SFSSQL especifica um conjunto de tipos de geometrias vetoriais, operações topológicas e operações métricas. Define também uma tabela de meta-dados. Para garantir que os meta-dados permaneçam consistentes, certas operações são chamadas por meio de *procedures* especiais do OpenGIS. Estas duas tabelas são respectivamente: SPATIAL_REF_SYS que armazena dados sobre cada sistema de referenciamento espacial utilizado no banco; e GEOMETRY_COLUMNS que armazena as colunas geométricas das tabelas com feições. As definições das tabelas são descritas abaixo.

Colunas da tabela SPATIAL_REF_SYS:

a) **SRID** - Um valor inteiro que exclusivamente identifica o Spatial Referecing System (SRS) dentro do banco de dados;

b) **AUTH_NAME**: Nome do corpo de padrões que está sendo citado para o sistema de referência;

c) **AUTH_SRID**: ID do SRID como definido pela autorizado citada no AUTH_NAME;

d) **SRTEXT**: representação WKT do SRS.

Colunas da tabela GEOMETRY_COLUMNS:

e) **F_TABLE_CATALOG**, **F_TABLE_SCHEMA**, **F_TABLE_NAME**: NAME é nome completamente qualificado da tabela contendo a coluna

geométrica. Segundo o Manual do PostGIS (Refractions, 2007) CATALOG e SCHEMA são padrões importados do Oracle;

f) **F_GEOMETRY_COLUMN**: nome da coluna geométrica na tabela;

g) **COORD_DIMENSION**: dimensão do espaço (2D ou 3D);

h) **SRID**: identificador do SRS usado para a coordenada geométrica nesta tabela;

i) **TYPE**: tipo do objeto espacial.

Segundo Câmara et al (2005), a representação dos dados espaciais podem seguir dois modelos, chamados SQL-92 e SQL-92 com Tipos Geométricos. O primeiro modelo utiliza uma tabela para representar os atributos espaciais, o segundo utiliza tipos abstratos de dados específicos da geometria.

6.2.4.4 Bibliotecas adicionais (GEOS e Proj4)

O suporte aos operadores espaciais no PostGIS é fornecido por meio da integração com a biblioteca Geometry Engine Open Source (GEOS). Esta biblioteca é uma tradução da API Java Java Topology Suite (JTS) para a linguagem de programação C++. JTS é uma implementação de operadores espaciais que seguem o padrão SFSSQL (CÂMARA et al, 2005).

A biblioteca Proj4 fornece as potencialidades de transformação das coordenadas. A coluna de PROJ4TEXT contém a String de definição de coordenada Proj4 para um SRID particular.

6.2.4.5 Índices Espaciais

Como já explicado anteriormente, o PostgreSQL provê vários tipos de índices (B-Tree, Hash, R-Tree e GiST), porém, no PostGIS, apenas os índices GiST e B-Tree podem indexar os dados espaciais (REFRACTIONS, 2007).

B-Trees são usados para consultas de igualdade e diferença em dados que podem ser ordenados ao longo de um eixo. O PostGIS irá utilizar este índice quando forem usados os operadores não espaciais (mas que são usados para comparar geometrias):

a) < (menor);

b) <= (menor igual);

c) = (igual);

d) >= (maior igual);

e) > (maior).

Uma grande deficiência do B-Tree é que o tamanho máximo do objeto a ser indexado não pode exceder o tamanho de 8K (8192 Bytes), e segundo Refractions (2007) dados de SIG geralmente são maiores que isso.

O índice GiST no PostGIS utiliza uma implementação de R-Tree. R-Tree podem abrir dados em retângulos, sub-retângulos, sub-sub-retângulos, e continuamente. R-Tree é usado como índice espacial no PostgreSQL, porém não é tão robusta como um índice GiST do PostGIS (Refractions, 2007). GiST pode ser usado para diversos tipos de dados, inclusive geográficos e sua grande vantagem segundo a Refractions (2007), é que praticamente não há limite para o tamanho do objeto a ser indexado. O PostGIS irá utilizar este índice quando forem utilizados os operadores espaciais:

a) << (está estritamente a esquerda de);

b) &< (não estende a direita de);

c) &>(não estende a esquerda de);

d) >> (está estritamente a direita de);

e) <<| (está estritamente abaixo);

f) &<| (não estende abaixo);

g) |&> (não estende acima);

h) |>> (está estritamente sobre);

i) @> (contém);

j) <@ (contém dentro ou em);

k) ~= (igual como) e && (sobrepõe).

6.3 PROBLEMAS ENCONTRADOS

Nenhum problema foi encontrado no armazenamento dos bancos de dados utilizados para o teste. Porém ocorreram problemas na criação dos métodos de acesso espaciais, especialmente na criação do índice B-Tree. Segundo a seção anterior, o método B-Tree possui limitações no armazenamento de dados. A solução encontrada foi recompilar o código-fonte do PostgreSQL, aproveitando o fato de que o mesmo é de código aberto e adaptá-lo para um maior poder de armazenamento nos índices B-Tree.

Após pesquisas na documentação do PostgreSQL foi possível aumentar o poder de armazenamento das B-Trees. Os seguintes arquivos tiveram trechos de código modificados:

a) **pg_config_manual.h**: esta biblioteca contém algumas configurações gerais do PostgreSQL. Neste arquivo a constante BLCKSZ, que é o tamanho do bloco do disco e também limita o tamanho dos registros, foi modificada. O valor inicial que era 8192 *bytes* foi alterado para 32768 *bytes* (máximo permitido);

b) **nbtree.h**: biblioteca que contém as funções do método B-Tree. Neste arquivo a constante "BTMaxItemSize" foi alterada para armazenar os itens dos índices em todos os espaços disponível nas páginas;

c) **indextuple.c**: este arquivo contém as rotinas e funções dos índices e vários utilitários para os registros. Neste código a função que limita o índice a um número pré-definido de registros foi comentada a fim de ignorar esta limitação. Esta função está localizada na linha 00165.

Após estas modificações o PostgreSQL ficou pronto para ser compilado no sistema operacional utilizado para os testes.

6.4 BANCOS DE DADOS UTILIZADOS PARA OS TESTES

Para a realização dos testes de eficiência dos métodos, foram utilizados três bancos de dados geográficos encontrados gratuitamente na internet: o projeto GeoMinas, o projeto Brasil Retis 2000 e o SIG da cidade de Tampa (EUA). Todos estes bancos encontram-se no formato ESRI[1] *Shape* que contém as coordenadas espaciais em formato *raster* e os metadados dos objetos espaciais. As sub-seções abaixo dão uma breve descrição dos dados compostos em cada banco de dados.

6.4.1 GeoMinas

O Programa Integrado de Uso da Tecnologia de Geoprocessamento pelos Órgãos do Estado de Minas Gerais (GeoMinas) surgiu com o objetivo de definir uma política de uso da tecnologia de geoprocessamento visando suprir a falta e a inexistência de mapas e

[1] ESRI é uma empresa que atua no ramo de SIG situada em San Diego na Califórnia, Estados Unidos.

informações para a tomada de decisões (Governo de Minas Gerais, 1996). O banco de dados utilizado para os testes possui vinte e cinco tabelas com tamanho total de quinze megabytes (15MB). A Tabela 3 contém a descrição destas tabelas.

Tabela 3. Lista das tabelas contidas no banco de dados GeoMinas

Nome tabela	Descrição	Geometria
br_carta_inter:	Carta Internacional do Mundo ao Milionésimo	Polígono
br_sercart_250:	BRASIL - Série Cartográfica 1:250.000	Polígono
br_sercart_500:	BRASIL - Série Cartográfica 1:500.000	Polígono
brasil:	Brasil	Polígono
macro_reg:	Macrorregiões de Planejamento	Polígono
meso_reg:	Mesorregiões Geográficas do IBGE	Polígono
mg:	Minas Gerais	Polígono
mg_adm_reg:	Regiões Administrativas - Lei 11962 30/10/95	Polígono
mg_asso_reg:	Associações Microrregionais da Seam	Polígono
mg_def_hidr:	Deficiências Hídricas ANUAIS	Polígono
mg_exced_hidr:	Excedentes Hídricos ANUAIS	Polígono
mg_hidro:	Hidrografia	Polígono
mg_ind_hidri:	Índices Hídricos Anuais	Polígono
mg_malha_aerea:	Aerovias	Linha
mg_malha_ferrov:	Ferrovias	Linha
mg_malha_hidro:	Hidrovias	Linha
mg_malha_rodov:	Rodovias	Linha
mg_micro_reg	Microrregiões Geográficas do IBGE	Polígono
mg_municipios	Divisão Político-Administrativa - 853 municípios	Polígono
mg_precipit	Precipitação Total Anual	Polígono
mg_sedes_muni	Sedes Municipais - 853 municípios - 1996	Polígono
mg_sercart	Séries Cartográficas de Minas Gerais	Polígono
mg_sercart_100	Série Cartográfica 1:100.000 de Minas Gerais	Polígono
mg_sercart_50	Série Cartográfica 1:50.000 de Minas Gerais	Polígono
mg_temp	Temperatura Média Anual	Polígono

Fonte: GOVERNO DE MINAS GERAIS (1996)

6.4.2 Brasil Retis 2000

O Grupo Retis de Pesquisa é formado por bolsistas de iniciação cientifica, mestrandos, mestres, doutorandos, doutores e diversos pesquisadores. A linha de pesquisa do grupo tem quatro objetivos principais. O primeiro é o desenvolvimento da investigação geográfica segundo conceitos oriundos da teoria dos sistemas complexos evolutivos e da

teoria de redes. O segundo é processar informações espaciais para subsidiar políticas públicas. O terceiro objetivo é explorar o instrumental oferecido pelo SIG para produção de mapas temáticos e análise espacial. O quarto e último objetivo é oferecer material didático e pedagógico para professores e alunos na área da geografia e ciências conexas. O grupo atua no departamento de geografia da Universidade Federal do Rio de Janeiro (UFRJ). Este grupo disponibiliza uma base de dados gratuita para consulta, chamada Brasil Retis 2000 que contém informações sobre o Brasil e algumas informações sobre a América Latina. Foram utilizadas 22 tabelas desta base de dados totalizando cinqüenta e dois megabytes (52MB) de informação, a descrição destas tabelas está na Tabela 4 (GRUPO DE PESQUISA RETIS, 2002).

Tabela 4. Lista das tabelas contidas no banco de dados Brasil Retis 2000

Nome tabela	**Descrição**	**Geometria**
aerofacp_point	Aeroportos da América Latina	ponto
ascoastl_line	Costa litorânea da América Latina	linha
aselevp_point	Principais relevos da América Latina	ponto
aspolbndl_line	Divisões estaduais dos países da América Latina	linha
braeroportointer	Aeroportos internacionais do Brasil	ponto
braldeiaindio	Principais aldeia indigenas do Brasil	ponto
brcampopouso	Campos de pouso do Brasil	ponto
brcapitaluf	Capitais estaduais	ponto
brcidade	Cidades do Brasil	ponto
brlocalidades	Localidades do Brasil	ponto
brmuni	Cidades do Brasil	polígono
brnucleo	Principais núcleos do Brasil	ponto
brpostoindio	Postos indígenas	ponto
brpovoado	Povoados	ponto
brprefixorodo	Rodovias	ponto
brprefixorodofed	Rodovias Federais	ponto
brterraindigena	Terras indígenas	polígono
brviarioempav	Rodovias em pavimentação	linha
brviarionpav	Rodovias não pavimentas	linha
brviariooutras	Outras rodovias	linha
brviariopav	Rodovias pavimentas	linha
brvila	Vilas do Brasil	ponto

Fonte: GRUPO DE PESQUISA RETIS (2002)

6.4.3 Tampa

A cidade de Tampa que fica situada no estado da Flórida nos Estados Unidos disponibiliza gratuitamente uma base de dados geográficos para quaisquer fins de consulta ou pesquisa (City of Tampa, 2007). Oito tabelas foram utilizadas desta base de dados que possuem o tamanho total de setenta e oito megabytes (78MB). A descrição das tabelas pode ser conferida na Tabela 5.

Tabela 5. Lista das tabelas contidas no banco de dados Tampa

Nome tabela	**Descrição**	**Geometria**
blocks1990	Divisão da cidade em blocos	polígono
council	divisão conselhos municipais	polígono
evaczone	Zonas de evacuação	polígono
floodzne	Zonas de risco de inundação	polígono
hydrants	Hidrantes da cidade	ponto
nodes	Nós de ligação	ponto
roads	Malha viária	linha
zipcodes	Divisão dos códigos postais	polígono

Fonte: CITY OF TAMPA (2007)

6.5 APLICAÇÃO DESENVOLVIDA

A fim de testar de forma eficaz os métodos de acesso espaciais, foi desenvolvida uma aplicação para a realização dos testes. Um diagrama de atividades foi elaborado para representar as funcionalidades da aplicação (Figura 49).

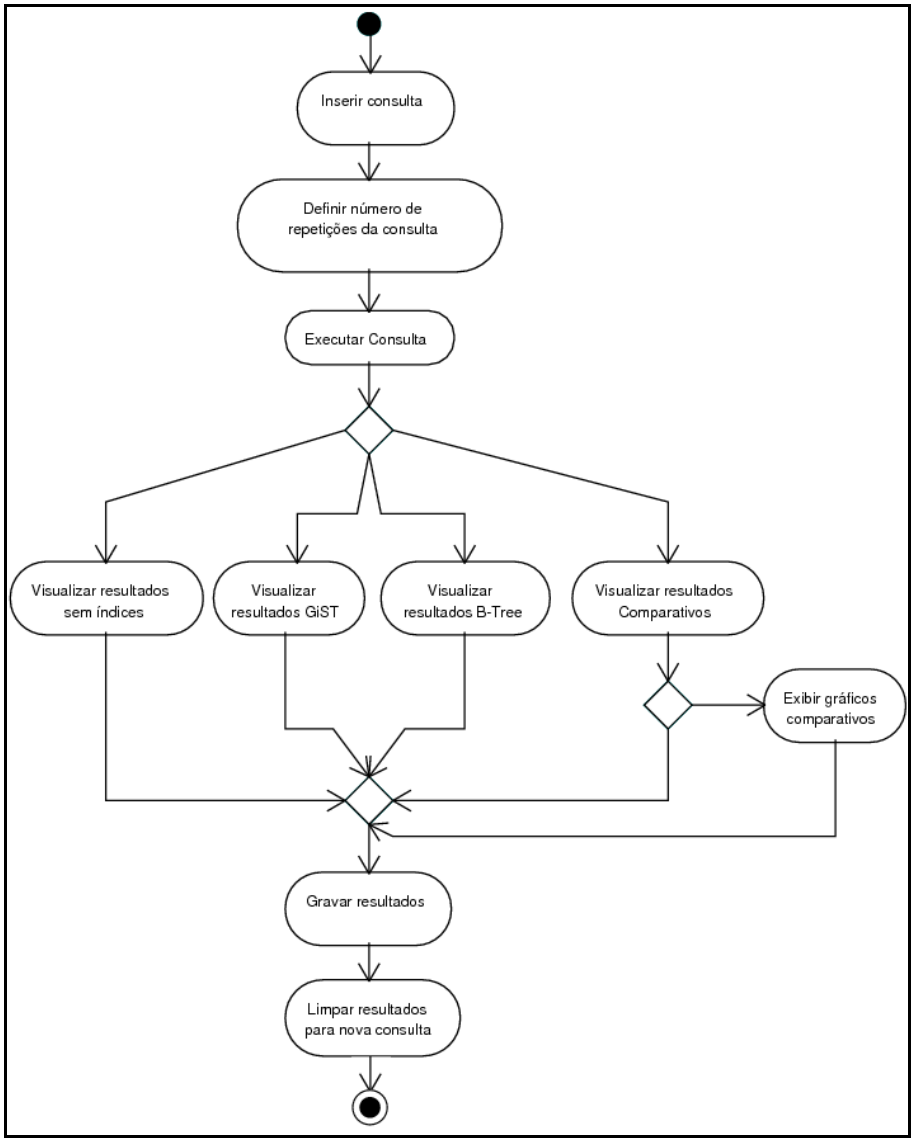

Figura 49. Diagrama de atividade da aplicação desenvolvida

A ferramenta NetBeans versão 5.5 que utiliza o JDK versão 1.6.0.1 foi utilizada para o desenvolvimento. A linguagem Java foi escolhida, pois possui um alto poder de abstração, gerando assim maior facilidade na implementação das funções do software e também por possuir uma biblioteca (JDBC) que garante fácil acesso ao banco de dados. O NetBeans surgiu como idéia, pois permite um desenvolvimento rápido. É uma ferramenta de Desenvolvimento Rápido de Aplicações (RAD) que permite uma fácil associação de um componente da interface ao código fonte. Um fator que contribuiu para a escolha desta ferramenta, é que tanto ela quanto a linguagem de programação utilizada, são softwares livres (gratuitos).

A principal operação do sistema é a execução da consulta, pois a partir desta, todas as informações podem ser analisadas. O usuário entra com uma consulta SQL e pode escolher o número de repetições desta consulta a fim de ter um resultado mais equilibrado.

Após executar a consulta quatro abas diferentes podem ser acessadas. A primeira exibe o resultado da consulta sem a utilização de índices, a segunda exibe os resultados utilizando o método GiST, o terceiro exibe com o método B-Tree e a última exibe uma tabela comparativa do tempo em milissegundos gastos pelas consultas em cada tipo de índice e o tempo médio para criação dos índices. Nesta aba também há a opção de gerar gráficos de barras, de linhas xy e de pizza para uma melhor abstração dos resultados. Todos os resultados podem ser gravados em arquivos de texto e de imagem para eventual análise.

6.6 DESCRIÇÃO DOS TESTES

Todos os testes foram realizados em um computador de configurações:

a) processador AMD *Athlon* XP 2600+ com *clock* aproximado de 1950 *megahertz* (MHz);

b) 1 *gigabyte* (GB) de memória principal;

c) disco rígido com capacidade de 120GB.

A primeira etapa dos testes consistiu na instalação do sistema operacional GNU/Linux Fedora Core 6, *kernel* 2.6.18-1.2798.fc6, com sistema de arquivos *ext3*. A versão utilizada para o SGDB PostgreSQL foi a 8.2.4 e para extensão PostGIS foi a versão 1.2.1. Ambos foram instalados com as suas opções padrões, exceto o PostgreSQL que teve trechos de seu código modificado, como descrito na seção 6.3 e tendo que ser compilado e preparado para a instalação. Após a instalação dos sistemas os bancos de dados citados na seção 6.4 foram exportados para o PostgreSQL, ficando prontos para a realização dos testes.

Os testes utilizando a aplicação desenvolvida foram feitos baseados na seguinte metodologia:

a) dez consultas diferentes para cada banco de dados;

b) as primeiras cinco consultas envolvendo operadores do método (1-5) GiST e as cinco últimas (6-10) envolvendo operadores do método B-Tree;

c) cada consulta foi repetida dez vezes, para se obter um resultado equilibrado.

Todos os resultados foram armazenados em arquivos de texto e de imagem para a análise comparativa final dos métodos de acesso espacial.

7 RESULTADOS DOS TESTES

Para a apresentação dos resultados foram selecionados aqueles que apontam maior discrepância de desempenho entre os diferentes MAE. A análise foi dividida basicamente em três partes: inicialmente foi analisado o desempenho em cada base de dados, logo após foi analisado o desempenho em relação aos operadores espaciais e a geometria e por último será realizado um comparativo final.

7.1 RESULTADO GEOMINAS

Conforme os resultados obtidos pela aplicação, o GiST foi o MAE que obteve o melhor resultado geral no banco de dados GeoMinas, tendo um ganho muito superior em relação aos outros métodos, como exibido no gráfico da Figura 50.

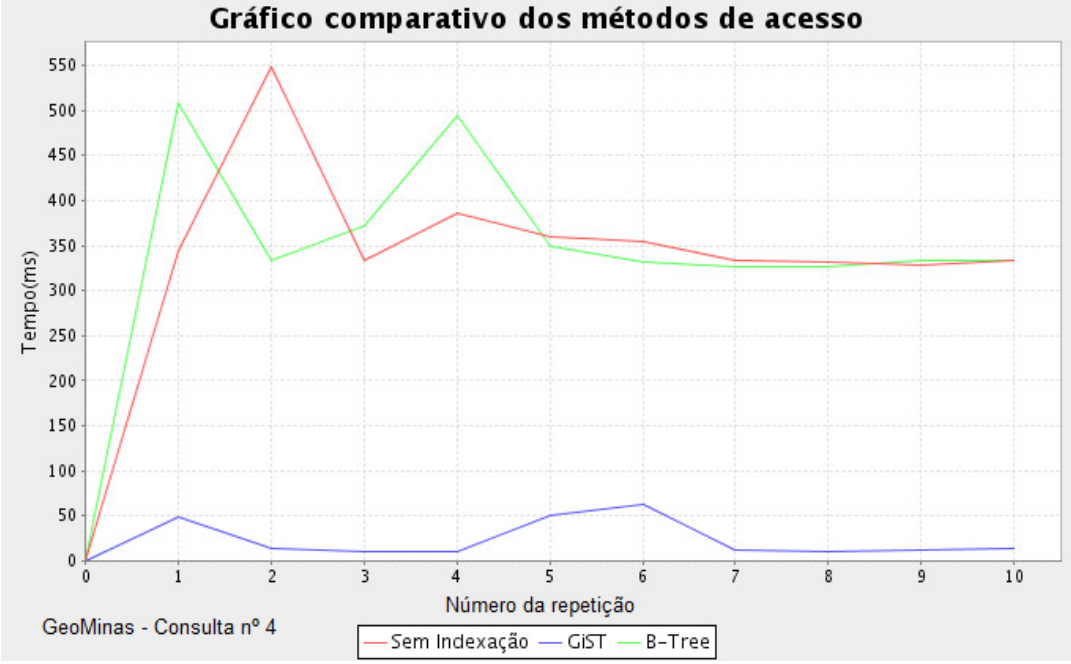

Figura 50. Gráfico de linhas do desempenho do método GiST

Na comparação geral, testes com índices e sem índices, o método GiST teve 196,66% de vantagem contra 74,69% do B-Tree. As consultas em que o método GiST foi mais eficiente somam 142,73% de ganho de desempenho, contra 40,53% do método B-Tree.

Quanto à utilização dos operadores específicos de cada método, operadores estes citados na seção 6.2.4.5, novamente o método GiST obteve os melhores resultados (Tabela 6).

Tabela 6. Ganhos em percentagem dos MAE no banco de dados GeoMinas

Tipo de ganho	GiST	B-Tree
Ganho com a utilização do índice (%)	196,66	74,69
Ganho de desempenho índice x índice (%)	142,73	40,53
Ganho de desempenho com operadores próprios (%)	285	81,02

7.2 RESULTADO BRASIL RETIS 2000

No banco de dados Brasil Retis 2000 o método GiST foi novamente o mais eficiente. Um exemplo disto está no gráfico da Figura 51, onde na consulta de número 5 a margem de ganho foi de quase 630%.

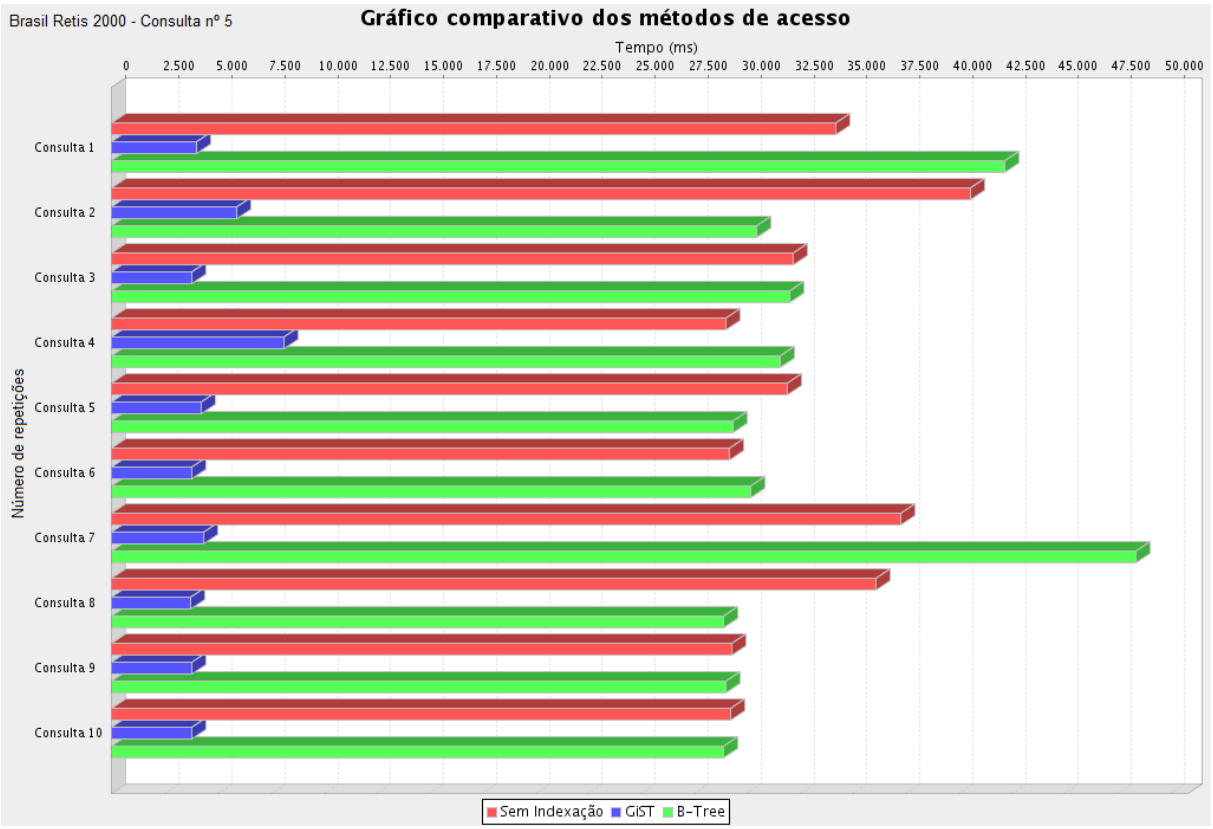

Figura 51. Gráfico de barras exibindo o desempenho do método GiST

De acordo com a Tabela 7, no quesito desempenho com índice e sem índice, GiST teve um ganho de 373,52% e B-Tree 59,91%. Na comparação índice com índice, GiST foi o

método mais eficiente em seis das dez consultas executadas, com desempenho de 230,99% contra 2,41% do método B-Tree em suas consultas mais eficientes. No ganho em relação aos operadores próprios de cada MAE, o método GiST obteve índices muitos altos, 459,55% contra 2,42% do método B-Tree.

Tabela 7. Ganhos em percentagem dos MAE no banco de dados Brasil Retis 2000

Tipo de ganho	GiST	B-Tree
Ganho com a utilização do índice (%)	373,52	59,91
Ganho de desempenho índice x índice (%)	230,99	2,41
Ganho de desempenho com operadores próprios (%)	459,55	2,42

7.3 RESULTADO TAMPA

Nos resultados dos testes realizados no banco de dados Tampa, o método GiST foi novamente o método com os resultados mais positivos, o ganho nas consultas foi de até 329% (Figura 52) em relação ao método B-Tree.

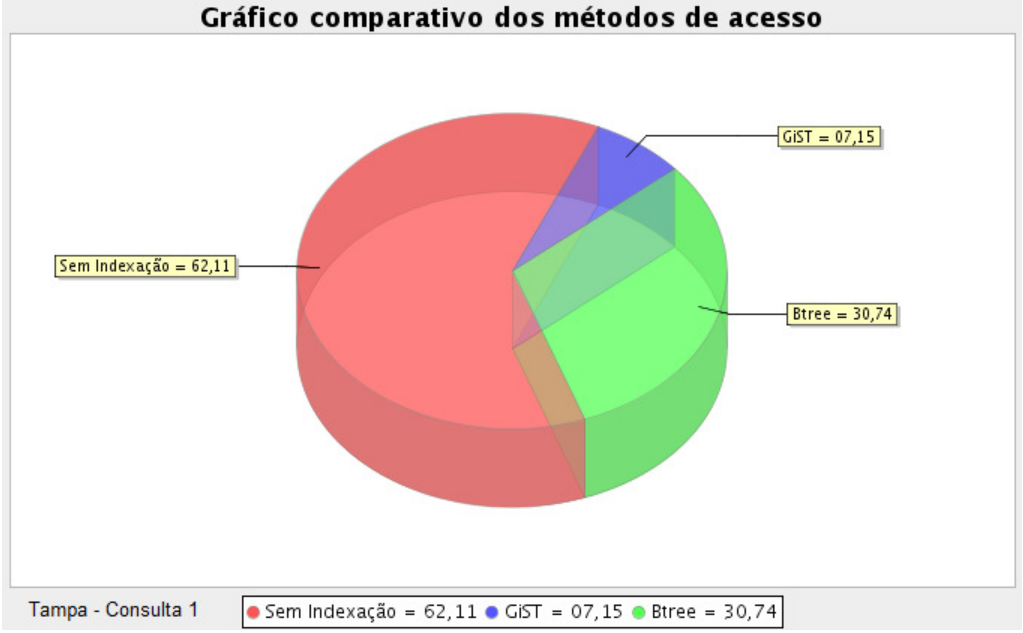

Figura 52. Gráfico exibindo o tempo em percentagem gasto por cada método

Os resultados de comparação sem a utilização de índice mostraram um ganho de 137,98% para o método GiST e 32,30% para o método B-Tree (Tabela 8). GiST foi superior

em cinco das dez consultas executadas, sendo que nas consultas em que ele foi superior houve um ganho de 71,32% contra 14,66% do método B-Tree em seus melhores resultados. Na utilização dos operadores específicos de cada método GiST teve um desempenho de 141,06% contra 28,74% de B-Tree chegando a ser melhor em algumas consultas utilizando o operador específico das B-Trees.

Tabela 8. Ganhos em percentagem dos MAE no banco de dados Tampa

Tipo de ganho	**GiST**	**B-Tree**
Ganho com a utilização do índice (%)	137,98	32,30
Ganho de desempenho índice x índice (%)	71,33	14,67
Ganho de desempenho com operadores próprios (%)	141,06	28,74

7.4 TEMPO DE CRIAÇÃO DOS ÍNDICES

O tempo para criação dos índices também foi analisado. A média total foi calculada somando todos os intervalos de tempo e dividindo pelo número de consultas em cada banco de dados (Tabela 9).

Tabela 9. Média do intervalo de tempo para criação dos MAE (em ms)

Banco de dados	**GiST**	**B-Tree**
GeoMinas	1108	4151
Brasil Retis 2000	11953	12686
Tampa	11885	12475

Neste atributo o método GiST foi mais uma vez o mais eficiente em todos os bancos de dados, chegando a ser até 274,64% vezes mais rápido (Figura 53), como no banco de dados GeoMinas.

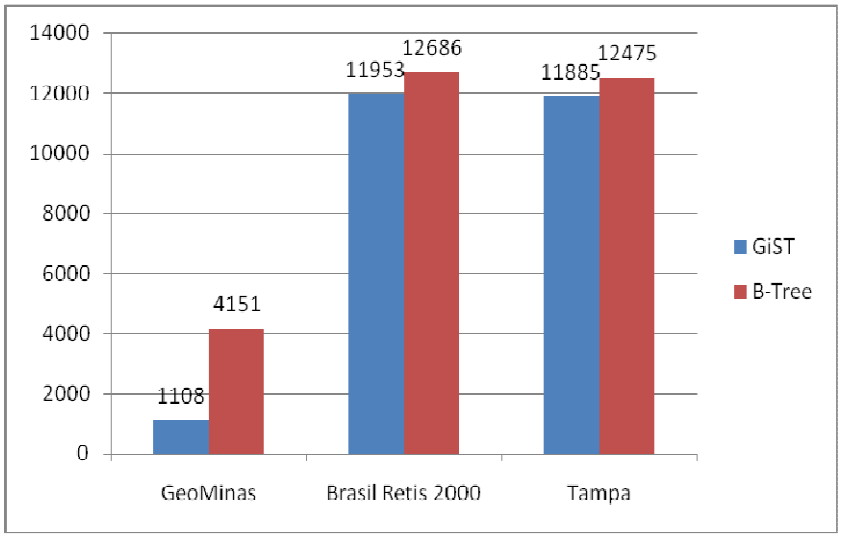

Figura 53. Gráfico exibindo o intervalo médio de tempo para criação dos índices

7.5 DESEMPENHO EM RELAÇÃO AOS OPERADORES ESPACIAIS

O desempenho dos MAE foi maior quando utilizados os operadores padrões de cada método descritos na seção 6.2.4.5. O método GiST além de ter a maior vantagem na utilização de seus métodos também teve o melhor desempenho em relação a algumas funções espaciais do PostGIS como *crosses*, *touches*, *overlaps* e *intersects*. B-Tree conseguiu alguns bons resultados com as funções *length*, *distance* e *area*. Em algumas consultas, o método GiST foi mais eficiente que o B-Tree, mesmo utilizando os operadores padrões do último método.

7.6 DESEMPENHO EM RELAÇÃO À GEOMETRIA

O desempenho dos métodos em relação à geometria utilizada (ponto, linha, polígono) pode ser verificado na Tabela 10.

Tabela 10.Percentual em relação às consultas onde os MAE foram mais eficientes

Banco de dados/Geometria	Ponto	Linha	Polígono
GiST	25%	28,58%	68,42%
B-Tree	75%	71,42%	31,58%

Neste caso pode-se observar a única vantagem vista até agora sobre o método B-Tree, onde o mesmo teve um melhor desempenho no caso em que a geometria envolvida na consulta foi do tipo ponto ou linha. O método GiST teve o melhor resultado em polígonos, que formam a maior parte dos bancos de dados geográficos utilizados e também foi o mais eficiente em todas as consultas que havia cruzamento de geometrias, ou seja, consultas de polígonos com linhas, pontos com linhas e pontos com polígonos. Mesmo assim o método B-Tree apresentou pouco ganho de desempenho nas suas geometrias mais eficientes.

7.7 RESULTADO FINAL

Tendo em vista os resultados apresentados nas seções anteriores, prova-se que o método GiST foi mais eficiente que o método B-Tree. Os fatores que contribuíram para esta conclusão foram respectivamente:

a) **melhor desempenho em todos as bases de dados**: o método GiST teve uma média total de ganho de desempenho de 148,35% em relação a B-Tree em todos as bases de dados geográficos utilizados;

b) **ganho de tempo na criação dos índices espaciais**: tempo médio de criação dos índices GiST foi de 8315 milissegundos contra 9770 milissegundos do método B-Tree. Isto não aparenta grande desempenho, porém no banco de dados GeoMinas que é composto em sua maioria pelo tipo de geometria polígono o tempo de criação dos índices GiST foi de 1108 milissegundos contra 4151 milissegundos do método B-Tree, ou seja, um ganho de 274,64%;

c) **maior eficiência em relação aos operadores geométricos**: GiST teve um melhor desempenho em relação aos operadores descritos na seção 6.2.4.5,

chegando a ser superior a B-Tree em consultas utilizando operadores deste último, como na consulta número 9 do banco de dados Brasil Retis 2000 (Tabela 11) em que foi utilizado o operador “>” (padrão do B-Tree) e mesmo assim GiST teve um desempenho maior.

Tabela 11. Resultado da consulta número 9 no banco de dados Brasil Retis 2000

Tipo de ganho	**GiST**	**B-Tree**
Ganho com a utilização do índice (%)	8,62%	Sem ganho
Ganho de desempenho índice x índice (%)	12,11%	Sem ganho
Ganho de desempenho com operadores padrões B-Tree	26,00%	Sem ganho

O único quesito em que B-Tree teve um desempenho superior a GiST, foi em relação as consultas realizadas em geometrias do tipo ponto e linha, mas utilizá-lo sozinho como índice em um banco de dados geográficos seria perda de desempenho. A modificação do núcleo do código-fonte do PostgreSQL para aumentar a sua capacidade para o armazenamento de polígonos não tem ganhos de desempenho aparentes, apenas permite um maior poder de armazenamento para o método B-Tree.

O método GiST foi mais estável em relação as variações no tempo das consultas, ou seja, além de se manter superior em relação ao desempenho na duração da consulta, apresentou estabilidade demonstrando novamente a sua superioridade na indexação dos dados. Um exemplo disto pode ser observado na Figura 54 onde as consultas no banco de dados sem a utilização de índices e com o método B-Tree tiveram grandes variações e o método GiST se comportou de maneira estável. O método B-Tree também apresentou grande estabilidade em grande parte das consultas, porém não na mesma proporção do método GiST.

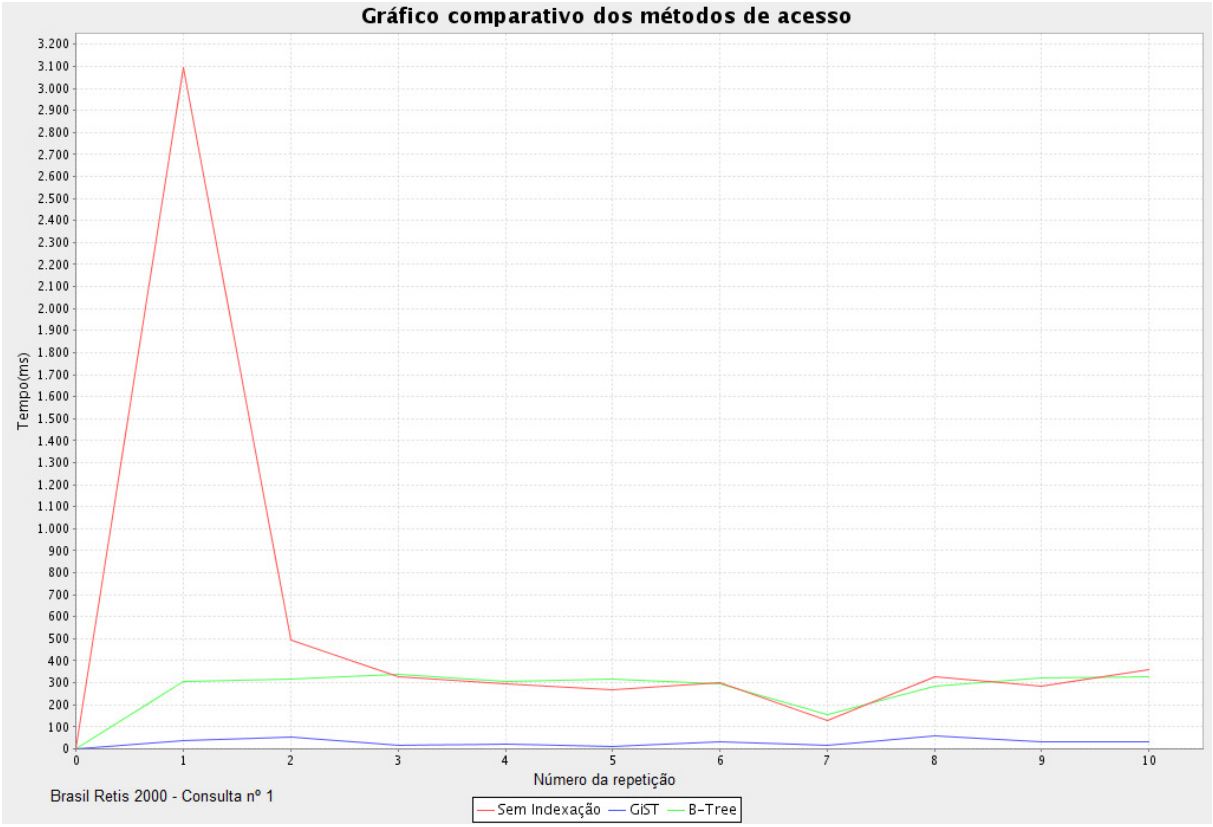

Figura 54. Exemplo da estabilidade do método GiST

De acordo com os resultados obtidos, o método B-Tree pode ser utilizado em conjunto com o GiST apenas em tabelas onde existam pontos e linhas para indexar estes tipos de dados, porém a só utilização do método GiST já garante um total desempenho com quaisquer tipos de operadores e de geometrias em um banco de dados geográficos.

CONCLUSÃO

Este trabalho apresentou um estudo sobre os métodos de acesso espaciais, sendo que o objetivo primordial foi definir quais são os métodos mais eficientes, utilizando como ferramenta o PostGIS. Uma extensa pesquisa foi realizada sobre os bancos de dados de geográficos e os sistemas a que estes pertencem, os Sistemas de Informações Geográficas.

Três diferentes bases de dados geográficos foram utilizadas para os testes sendo que para o SGDB PostgreSQL ficar com uma capacidade maior de armazenamento, trechos de seu código-fonte foram modificados satisfazendo as necessidades requeridas para a utilização do método de acesso B-Tree. Uma aplicação gráfica foi desenvolvida para calcular o tempo gasto por cada consulta executada em cada método, ou seja, para armazenar os resultados para uma análise comparativa.

A análise do desempenho apontou o método GiST como o mais eficiente, sendo este superior em quase todos os testes, onde apresentou uma média de ganho de desempenho sobre o método B-Tree de 148,35%. Nas consultas que envolviam os tipos de geometria ponto e linha, B-Tree se saiu superior, porém não com grande vantagem se comparado com os resultados de GiST. O método B-Tree chegou a ser inferior em algumas consultas onde foram utilizados seus operadores padrões. O trabalho operacional gasto na configuração no código-fonte do PostgreSQL para deixar a implementação de B-Tree mais robusta também é mais um fator que se coloca contra ele.

O método GiST pode ser usado por si só para indexar um bancos de dados geográficos com diferentes tipos de geometrias, porém para uma configuração perfeita o método B-Tree pode ser eventualmente usado para indexar os tipos de geometria linha e ponto que tiveram resultados positivos com este método.

Infelizmente este trabalho de análise comparativa não é totalmente conclusivo, pois há ainda muitas metodologias diferentes para se testar o desempenho dos métodos de acesso espacial. Como trabalhos futuros sugerem-se:

a) análise dos métodos de acesso espacial em diferentes plataformas;

b) análise dos métodos de acesso espacial em diversos sistemas operacionais;

c) análise dos métodos de acesso espacial em sistemas multi-usuários.

REFERÊNCIAS

BRITO, Jorge Luís S.; ROSA, Roberto. **Introdução ao Geoprocessamento.** Uberlândia: Universidade Federal de Uberlândia, 1996.

CÂMARA, Gilberto et al. **Bancos de Dados Geográficos**. Curitiba: Mundo Geo, 2005.

CIFERRI, Rodrigo Ricardes; SALGADO, Ana Carolina. Métodos de Acesso Espaciais em Termos da Distribuição Espacial dos Dados. Workshop Brasileiro de Geoinformática (GEOINFO 2001). Rio de Janeiro, 2001. Anais. Local: Instituto Militar de Engenharia, 2001. Disponível em: **www.geoinfo.info/geoinfo2001/papers/145ciferri.pdf**. Acesso em: 22 de setembro de 2006.

CITY OF TAMPA. Department of Geographic Information Systems. Disponível em https://www.tampagov.net/dept_Geographic_Information_Systems/index.asp**.** Acesso em: Junho/2007.

COMER, D. The ubiquitous B-Tree. **ACM Computing Surveys**, v. 11, n. 2, p. 121-137, 1979.

GOVERNO DE MINAS GERAIS. Projeto GeoMinas. Disponível em: http://www.geominas.mg.gov.br/. Acesso em: Junho/2007.

GRUPO RETIS DE PESQUISA. Base de dados espaciais. Disponível em: http://www.igeo.ufrj.br/gruporetis/sig/tiki-list_file_gallery.php?galleryId=1**.** Acesso em: Junho, 2007.

HELLERSTEIN, Joseph M.; NAUGHTON, Joseph M.; PFEFFER, Avi. **Generalized Search Trees for Database Systems**. Proc. 21ª International Conferency. on Very Large Data Bases, Zürich, Suiça. p. 562-573. 1995.

JANDL, Peter Jr.; **Java 5**: Guia de consulta rápida. São Paulo: Novatec Editora Ltda., 2006.

KEMPER, A. **Proseminar: Algorithms and Data Structures for Persistent Data**. Passau, Alemanha, 2001. Disponível em: **http://www.fmi.uni-passau.de/~neumann/proseminar/proseminar.pdf**. Acesso em: Março, 2007.

MAEDA, Vinícius; SALES, Ronaldo; SIMONATO, Thiago. Modelagem de dados espaciais. **SQL Magazine.** São Paulo, n. 21, p. 32-39. 2005.

MELLO, Ronaldo S.. Bancos de Dados Geográficos. Disponível em: http://www.inf.ufsc.br/~ronaldo/ine5342/bdg.pdf. Florianópolis, 2004. Acesso em: 22 Novembro 2006.

NETBEANS. NetBeans IDE Documentation. Disponível em**:** http://www.netbeans.org/. Acesso em: Junho/2007.

NEVES, Denise Lemes Fernandes. **PostgreSQL: Conceitos e Aplicações**. São Paulo: Érica, 2002.

NIEDERAUER, Juliano. **PostgreSQL:** guia de consulta rápida. São Paulo: Novatec Editora Ltda., 2004.

NONATO, Luis G. **Árvores B**. Disponível em: http://www.lcad.icmc.usp.br/~nonato/ED/B_arvore/btree.htm. São Paulo, 2002. Acesso em: Março, 2007.

OOI, Beng Chin, TAN, Kian-Lee. B-Trees: Bearing Fruits of All Kinds. **Proceedings of the 13th Australasian database conference - Volume 5**. Melbourne, Austrália, p. 13-20. 2002.

PARADEDA, Raul B. **Árvores B – B Trees**. Disponível em http://descartes.ucpel.tche.br/WFC/2002/apa_grupo3-ArvoresB.pdf. Pelotas, 2004. Acesso em: Março, 2007.

PAREDES, Evaristo A. **Sistema de informação geográfica - SIG**: princípios e aplicação (geoprocessamento). São Paulo: Érica, 1994.

POSTGRESQL. **PostgreSQL Documentation.** Disponível em: http://www.postgresql.org**/.** Acesso em: Junho, 2007.

RAMON, Fábio. **JDBC 2**: Guia de consulta rápida. São Paulo: Novatec Editora Ltda., 2000.

REFRACTIONS. **PostGIS Manual**. Disponível em: http://postgis.refractions.net/docs**/.** Acesso em: Fevereiro, 2007.

SILVA, Ardemirio de Barros. **Sistemas de Informações Geo-referenciadas.** Campinas: Editora da Unicamp, 1999.

SILVA, Rosângela. **Bancos de Dados Geográficos:** uma análise das arquiteturas dual (Spring) e integrada (Oracle Spatial). São Paulo: Editora USP, 2002.

TEIXEIRA, Amandio Luís de Almeida; CHRISTOFOLETTI, Antônio**. Sistemas de informação geográfica;** (dicionário ilustrado). São Paulo: Hucitec, 1997.

BIBLIOGRAFIA COMPLEMENTAR

CÂMARA, G.; CARVALHO, M.S.; DRUCK, S. **Análise Espacial de Dados Geográficos**. Brasília: EMBRAPA, 2004.

DAVIS Jr., C; Uso de vetores em GIS. **Fator GIS**, v. 21, n. 4, p. 22-23, 1997.

FERRARI, Roberto. **Viagem ao SIG**. Curitiba: Sagres Editora, 1997.

FIGUEIREDO, L. H.; CARVALHO, P. C. P. **Introdução à geometria computacional**. Rio de Janeiro: IMPA, 1991.

GARCIA-MOLINA, H.; ULLMAN, J. D.; WIDOM, J. **Implementação de sistemas de bancos de Dados**. Rio de Janeiro: Campus, 2001.

GOMES, Jonas; VELHO, Luiz. **Computação gráfica**. Rio de Janeiro: IMPA, 1998.

HORSTMANN, CAY S.; CORNELL GARY. **Core Java 2**: Volume I – Fundamentos. Rio de Janeiro: Altabooks, 2004.

PIAZZA, Rafael. **O sistema de informação geográfica, como ferramenta para monitoramento da cultura de arroz na região sul de Santa Catarina**. Criciúma: Universidade do Extremo Sul Catarinense, 2005.

RAMALHO, José Antônio A. **SQL: a linguagem dos bancos de dados.** São Paulo: Berkeley, 1999.

APÊNDICE A – TABELAS DOS RESULTADOS DAS CONSULTAS

BANCO DE DADOS GEOMINAS (RESULTADOS EM MILISSEGUNDOS)

Consulta 1: SELECT mg_malha_rodov.gid FROM mg_malha_rodov, mg_municipios WHERE mg_malha_rodov.the_geom && mg_municipios.the_geom AND intersects(mg_malha_rodov.the_geom, mg_municipios.the_geom) AND mg_municipios.nommuni = 'Diamantina'

Num. Repetição	Sem Indexação	GiST	B-Tree
1	73	23	29
2	33	18	19
3	22	19	20
4	20	19	20
5	24	18	19
6	19	18	20
7	19	18	20
8	20	19	19
9	20	19	21
10	28	18	20
Tempo médio de criação dos índices GiST:		473	
Tempo médio de criação dos índices B-Tree:		1990	

Consulta 2: SELECT m1.nommuni FROM mg_municipios m1, mg_municipios m2 WHERE m1.the_geom && m2.the_geom AND touches(m1.the_geom, m2.the_geom) AND (m1.nommuni <> 'Belo Horizonte') AND (m2.nommuni = 'Belo Horizonte')

Num. Repetição	Sem Indexação	GiST	B-Tree
1	60	10	10
2	21	10	10
3	11	9	9
4	11	9	9
5	19	9	10
6	11	11	10
7	11	10	10
8	9	9	10
9	10	9	11
10	10	9	9
Tempo médio de criação dos índices GiST:		552	
Tempo médio de criação dos índices B-Tree:		2138	

Consulta 3: SELECT mg_municipios.nommuni FROM mg_municipios, mg_malha_rodov WHERE intersects(buffer(mg_malha_rodov.the_geom,0.1),mg_municipios.the_geom) AND mg_malha_rodov.gid = 5

Num. Repetição	Sem Indexação	GiST	B-Tree
1	45.752	40.181	40.149
2	40.156	40.234	40.195
3	40.112	40.272	40.034
4	40.098	40.130	40.102
5	40.027	40.087	40.185
6	40.165	40.161	40.123
7	40.121	40.167	40.072
8	40.158	40.889	40.371
9	40.408	40.441	40.423
10	40.328	40.384	40.365
Tempo médio de criação dos índices GiST:		968	
Tempo médio de criação dos índices B-Tree:		2585	

Consulta 4: SELECT mg_malha_rodov.gid FROM mg_malha_rodov, mg_malha_hidro WHERE (mg_malha_rodov.the_geom && mg_malha_hidro.the_geom) AND within(mg_malha_rodov.the_geom, mg_malha_hidro.the_geom)

Num. Repetição	Sem Indexação	GiST	B-Tree
1	344	49	508
2	548	13	334
3	333	11	371
4	386	11	494
5	360	51	349
6	355	62	331
7	334	12	326
8	331	11	326
9	328	12	333
10	334	13	333
Tempo médio de criação dos índices GiST:		1243	
Tempo médio de criação dos índices B-Tree:		4740	

Consulta 5: SELECT mg_malha_ferrov.gid FROM mg_malha_ferrov, mg_malha_rodov WHERE mg_malha_ferrov.the_geom && mg_malha_rodov.the_geom AND crosses(mg_malha_ferrov.the_geom,mg_malha_rodov.the_geom)

Num. Repetição	Sem Indexação	GiST	B-Tree
1	3.046	2.905	2.764
2	2.749	2.762	2.752
3	2.755	2.747	2.763
4	2.755	2.743	2.749
5	2.755	2.777	2.754
6	2.758	2.796	4.137
7	3.311	3.214	2.713
8	2.716	2.691	2.704
9	2.708	2.761	2.717
10	2.710	2.704	2.715
Tempo médio de criação dos índices GiST:		1857	
Tempo médio de criação dos índices B-Tree:		7552	

Consulta 6: SELECT brasil.nomest,Area(brasil.the_geom) FROM brasil,mg WHERE Area(brasil.the_geom) < Area(mg.the_geom)

Num. Repetição	Sem Indexação	GiST	B-Tree
1	64	12	12
2	22	11	11
3	15	10	9
4	10	9	9
5	13	11	11
6	12	10	10
7	12	11	10
8	11	11	10
9	12	11	9
10	11	9	11
Tempo médio de criação dos índices GiST:		636	
Tempo médio de criação dos índices B-Tree:		3021	

Consulta 7: SELECT muni1.nommuni FROM mg_municipios muni1, mg_municipios muni2 WHERE Area(muni1.the_geom) >= Area(muni2.the_geom) AND muni1.nommuni <> 'Alagoa' AND muni2.nommuni = 'Alagoa'

Num. Repetição	Sem Indexação	GiST	B-Tree
1	852	23	24
2	18	18	18
3	19	18	17
4	19	18	17
5	19	17	18
6	95	17	17
7	19	17	17
8	19	16	17
9	20	88	17
10	20	17	16
Tempo médio de criação dos índices GiST:		1009	
Tempo médio de criação dos índices B-Tree:		2739	

Consulta 8: SELECT mg_micro_reg.nommic FROM mg_micro_reg, mg_malha_aerea WHERE distance(mg_micro_reg.the_geom, mg_malha_aerea.the_geom) > 4

Num. Repetição	Sem Indexação	GiST	B-Tree
1	499	542	482
2	484	500	483
3	487	546	506
4	484	501	480
5	486	497	480
6	600	501	479
7	492	498	482
8	608	500	482
9	494	569	480
10	496	500	479
Tempo médio de criação dos índices GiST:		1568	
Tempo médio de criação dos índices B-Tree:		5611	

Consulta 9: SELECT mr1.gid FROM mg_malha_rodov mr1, mg_malha_rodov mr2 WHERE length(mr1.the_geom) <= length(mr2.the_geom) AND mr2.nome_rodov = 'BR 040'

Num. Repetição	Sem Indexação	GiST	B-Tree
1	119	642	97
2	99	598	149
3	101	643	98
4	103	599	97
5	108	594	101
6	105	598	98
7	110	595	265
8	108	601	98
9	111	666	251
10	112	597	98
Tempo médio de criação dos índices GiST:		2130	
Tempo médio de criação dos índices B-Tree:		7886	

Consulta 10: SELECT mg_adm_reg.nomadm FROM mg_adm_reg, macro_reg WHERE Area(mg_adm_reg.the_geom) >= Area(macro_reg.the_geom)

Num. Repetição	Sem Indexação	GiST	B-Tree
1	97	30	29
2	40	31	30
3	30	31	31
4	35	30	32
5	37	29	30
6	29	30	29
7	30	30	30
8	29	31	29
9	33	32	29
10	34	30	29

Tempo médio de criação dos índices GiST: 640
Tempo médio de criação dos índices B-Tree: 3253

BANCO DE DADOS BRASIL_RETIS_2000 (RESULTADOS EM MILISSEGUNDOS)

Consulta 1: SELECT brviarionpav.nm_rodovia FROM brviarionpav,brmuni WHERE brviarionpav.the_geom && brmuni.the_geom AND intersects(brviarionpav.the_geom,brmuni.the_geom) AND brmuni.nome = 'Rio de Janeiro'

Num. Repetição	Sem Indexação	GiST	B-Tree
1	3.093	36	307
2	492	51	317
3	329	17	338
4	297	20	306
5	269	13	315
6	299	30	295
7	126	14	157
8	328	59	283
9	282	33	320
10	357	33	326

Tempo médio de criação dos índices GiST: 11577
Tempo médio de criação dos índices B-Tree: 11760

Consulta 2: SELECT brterraindigena.nm_nome FROM brterraindigena, brmuni WHERE brterraindigena.the_geom && brmuni.the_geom AND within(brterraindigena.the_geom, brmuni.the_geom)

Num. Repetição	Sem Indexação	GiST	B-Tree
1	6.952	800	6.688
2	6.642	802	6.631
3	6.653	762	6.756
4	6.722	775	6.661
5	6.667	753	6.622
6	6.677	779	6.688
7	6.718	761	6.697
8	6.717	1.055	11.467
9	6.977	791	6.566
10	8.973	902	6.520

Tempo médio de criação dos índices GiST: 27562
Tempo médio de criação dos índices B-Tree: 29985

Consulta 3: SELECT brmuni.nome FROM brmuni, brviariopav WHERE intersects(buffer(brviariopav.the_geom,0.1),brmuni.the_geom) AND brviariopav.gid = 14917

Num. Repetição	Sem Indexação	GiST	B-Tree
1	112.516	107.229	107.085
2	110.478	107.284	107.090
3	107.252	107.428	107.767
4	108.023	108.082	114.760
5	116.845	109.374	107.821
6	114.145	134.330	157.237
7	139.977	123.542	127.798
8	111.307	107.240	107.210
9	107.302	117.379	129.881
10	119.847	138.653	130.074

Tempo médio de criação dos índices GiST: 9171
Tempo médio de criação dos índices B-Tree: 8547

Consulta 4: SELECT b1.nome FROM brmuni b1, brmuni b2 WHERE touches(b1.the_geom, b2.the_geom) AND (b1.nome <> 'Curitiba') AND b1.the_geom && b2.the_geom AND (b2.nome = 'Curitiba')

Num. Repetição	Sem Indexação	GiST	B-Tree
1	262	13	22
2	33	13	22
3	25	12	24
4	27	14	22
5	26	13	22
6	22	13	23
7	109	15	22
8	22	12	23
9	24	63	23
10	23	12	22

Tempo médio de criação dos índices GiST: 8470
Tempo médio de criação dos índices B-Tree: 9303

Consulta 5: SELECT brmuni.nome FROM brmuni, ascoastl_line WHERE brmuni.the_geom && ascoastl_line.the_geom AND touches(brmuni.the_geom, ascoastl_line.the_geom) ORDER BY brmuni.gid

Num. Repetição	Sem Indexação	GiST	B-Tree
1	34.225	3.987	42.188
2	40.550	5.915	30.462
3	32.220	3.750	32.019
4	29.019	8.128	31.556
5	31.911	4.209	29.342
6	29.204	3.750	30.176
7	37.270	4.337	48.395
8	36.116	3.743	28.883
9	29.312	3.798	28.991
10	29.238	3.747	28.886

Tempo médio de criação dos índices GiST: 16562
Tempo médio de criação dos índices B-Tree: 18888

Consulta 6: SELECT brviarioempav.nm_sigla FROM brviarioempav, brviarionpav WHERE length(brviarioempav.the_geom) > length(brviarionpav.the_geom) GROUP BY brviarioempav.nm_sigla LIMIT 100

Num. Repetição	Sem Indexação	GiST	B-Tree
1	17.135	15.725	15.646
2	15.654	17.432	15.354
3	15.381	15.425	15.399
4	15.360	15.384	15.373
5	15.431	15.378	15.512
6	15.335	15.465	15.352
7	15.401	15.407	15.337
8	15.384	15.367	15.350
9	15.367	15.380	15.369
10	15.406	15.376	15.361
Tempo médio de criação dos índices GiST:			8641
Tempo médio de criação dos índices B-Tree:			8554

Consulta 7: SELECT brviarioempav.nm_sigla FROM brviarioempav, brviariopav WHERE length(brviarioempav.the_geom) = length(brviariopav.the_geom)

Num. Repetição	Sem Indexação	GiST	B-Tree
1	5.652	250	218
2	222	213	216
3	242	214	215
4	224	217	217
5	213	229	217
6	219	215	219
7	219	214	215
8	218	213	218
9	219	215	221
10	217	216	218
Tempo médio de criação dos índices GiST:			8855
Tempo médio de criação dos índices B-Tree:			8439

Consulta 8: SELECT braldeiaindio.nm_nome FROM braldeiaindio, brcampopouso WHERE distance(braldeiaindio.the_geom, brcampopouso.the_geom) > 30

Num. Repetição	Sem Indexação	GiST	B-Tree
1	488	350	349
2	442	344	351
3	343	341	328
4	347	341	344
5	461	344	343
6	396	341	348
7	490	340	356
8	541	354	365
9	542	475	345
10	361	336	336
Tempo médio de criação dos índices GiST:			8497
Tempo médio de criação dos índices B-Tree:			9612

Consulta 9: SELECT brcidade.nm_nome FROM brcidade, braldeiaindio WHERE distance(braldeiaindio.the_geom, brcidade.the_geom) < 10

Num. Repetição	Sem Indexação	GiST	B-Tree
1	795	624	837
2	859	1.310	617
3	589	952	591
4	677	710	607
5	715	596	647
6	815	757	531
7	892	1.146	971
8	755	619	581
9	3.688	495	553
10	983	482	544

Tempo médio de criação dos índices GiST: 9365

Tempo médio de criação dos índices B-Tree: 8966

Consulta 10: SELECT brvila.nm_nome FROM brviariooutras, brvila WHERE crosses(brviariooutras.the_geom, brvila.the_geom) AND length(brviariooutras.the_geom) > 0.2

Num. Repetição	Sem Indexação	GiST	B-Tree
1	11.488	10.562	11.028
2	11.993	13.147	12.503
3	15.155	14.540	12.552
4	12.159	11.028	12.768
5	15.051	13.337	12.297
6	15.655	14.543	16.446
7	15.456	13.044	14.632
8	14.610	11.617	17.263
9	12.353	12.194	14.987
10	13.082	12.119	16.929

Tempo médio de criação dos índices GiST: 10827

Tempo médio de criação dos índices B-Tree: 12803

BANCO DE DADOS TAMPA (RESULTADOS EM MILISSEGUNDOS)

Consulta 1: SELECT length(roads.the_geom) FROM roads, blocks1990 WHERE blocks1990.the_geom && roads.the_geom AND intersects(blocks1990.the_geom, roads.the_geom) AND blocks1990.name = '120570001 101'

Num. Repetição	Sem Indexação	GiST	B-Tree
1	2.153	46	193
2	187	45	193
3	187	44	193
4	194	47	195
5	195	43	192
6	193	44	191
7	198	43	191
8	192	45	192
9	192	46	192
10	193	44	190

Tempo médio de criação dos índices GiST: 7375

Tempo médio de criação dos índices B-Tree: 6954

Consulta 2: SELECT blocks1990.name FROM blocks1990, roads WHERE blocks1990.the_geom && roads.the_geom AND intersects(buffer(roads.the_geom,0.1),blocks1990.the_geom) AND roads.gid = 6126

Num. Repetição	Sem Indexação	GiST	B-Tree
1	169	18	66
2	56	17	68
3	69	17	68
4	53	17	68
5	53	19	66
6	59	17	66
7	53	18	69
8	60	17	67
9	54	18	69
10	58	18	79

Tempo médio de criação dos índices GiST: 7056

Tempo médio de criação dos índices B-Tree: 7276

Consulta 3: SELECT blocks1990.name FROM blocks1990, council WHERE blocks1990.the_geom && council.the_geom AND touches(blocks1990.the_geom, council.the_geom) AND blocks1990.gid < 30

Num. Repetição	Sem Indexação	GiST	B-Tree
1	244	168	169
2	213	203	167
3	180	277	170
4	193	172	259
5	210	171	200
6	194	204	195
7	178	186	182
8	176	176	161
9	160	164	164
10	164	164	164

Tempo médio de criação dos índices GiST: 7883

Tempo médio de criação dos índices B-Tree: 8254

Consulta 4: SELECT b1.name FROM blocks1990 b1, blocks1990 b2 WHERE b1.the_geom && b2.the_geom AND touches(b1.the_geom, b2.the_geom) AND (b1.name <> '120570001 101') AND (b2.name = '120570001 101')

Num. Repetição	Sem Indexação	GiST	B-Tree
1	1.250	43	59
2	252	175	503
3	45	43	71
4	46	49	75
5	46	117	105
6	53	69	188
7	53	53	84
8	57	82	81
9	63	46	79
10	49	53	77

Tempo médio de criação dos índices GiST: 9698
Tempo médio de criação dos índices B-Tree: 10605

Consulta 5: SELECT evaczone.gid FROM evaczone, council WHERE evaczone.the_geom && council.the_geom AND within(evaczone.the_geom, council.the_geom)

Num. Repetição	Sem Indexação	GiST	B-Tree
1	4.255	3.534	3.760
2	3.678	3.493	3.673
3	3.696	3.496	3.692
4	3.694	3.522	3.675
5	3.677	3.486	3.677
6	3.679	3.900	4.376
7	4.276	3.708	3.673
8	3.512	3.481	3.687
9	3.682	3.511	3.511
10	3.519	3.524	3.514

Tempo médio de criação dos índices GiST: 6296
Tempo médio de criação dos índices B-Tree: 7173

Consulta 6: SELECT b1.name FROM blocks1990 b1, blocks1990 b2 WHERE Area(b1.the_geom) = Area(b2.the_geom) AND b1.gid < 50 ORDER BY name LIMIT 5000

Num. Repetição	Sem Indexação	GiST	B-Tree
1	122	170	80
2	69	98	82
3	59	98	79
4	62	113	79
5	59	103	80
6	60	96	79
7	174	103	82
8	59	100	81
9	67	98	80
10	60	102	81

Tempo médio de criação dos índices GiST: 14202
Tempo médio de criação dos índices B-Tree: 14768

Consulta 7: SELECT r1.street FROM roads r1, roads r2 WHERE length(r1.the_geom) = length(r2.the_geom) ORDER BY street LIMIT 500

Num. Repetição	Sem Indexação	GiST	B-Tree
1	2.409	2.483	2.450
2	2.520	2.644	2.245
3	2.178	2.818	2.303
4	2.165	2.405	2.236
5	2.161	2.867	2.228
6	2.288	2.460	2.295
7	2.162	2.832	2.226
8	2.244	2.395	2.227
9	2.171	2.404	2.216
10	2.171	2.714	2.276

Tempo médio de criação dos índices GiST:	21488
Tempo médio de criação dos índices B-Tree:	22258

Consulta 8: SELECT r1.street FROM roads r1, roads r2 WHERE length(r1.the_geom) <= length(r2.the_geom) AND r1.gid = 2

Num. Repetição	Sem Indexação	GiST	B-Tree
1	1.196	653	559
2	791	567	655
3	607	496	522
4	620	640	480
5	761	568	526
6	891	562	531
7	885	577	569
8	3.923	604	522
9	1.076	524	529
10	3.706	522	475

Tempo médio de criação dos índices GiST:	8182
Tempo médio de criação dos índices B-Tree:	8709

Consulta 9: SELECT roads.street FROM roads, hydrants WHERE distance(roads.the_geom, hydrants.the_geom) < 1000 AND roads.gid < 10 ORDER BY street LIMIT 50

Num. Repetição	Sem Indexação	GiST	B-Tree
1	225	396	219
2	233	424	221
3	218	500	218
4	221	392	217
5	219	390	218
6	218	423	216
7	221	405	215
8	218	394	217
9	215	383	220
10	215	385	215

Tempo médio de criação dos índices GiST:	14683
Tempo médio de criação dos índices B-Tree:	15188

Consulta 10: SELECT evaczone.gid FROM evaczone, roads WHERE crosses(evaczone.the_geom, roads.the_geom) AND length(roads.the_geom) < 100

Num. Repetição	Sem Indexação	GiST	B-Tree
1	3.793	3.887	4.052
2	3.700	4.032	6.862
3	3.646	4.433	5.866
4	3.685	3.920	3.464
5	3.325	3.829	3.315
6	3.303	3.720	3.354
7	3.327	3.705	3.322
8	3.327	3.699	3.322
9	3.291	3.683	3.327
10	3.293	3.785	4.907

Tempo médio de criação dos índices GiST:	21988
Tempo médio de criação dos índices B-Tree:	23566