MINISTÉRIO DA CIÊNCIA E TECNOLOGIA
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

EXTENSÃO DO SGBD POSTGRESQL COM OPERADORES ESPACIAIS

Gilberto Ribeiro de Queiroz

Proposta de Mestrado em Computação Aplicada, orientada pelo Prof. Dr. João Argemiro C. Paiva e pelo Prof. Dr. Gilberto Câmara.

INPE
São José dos Campos
Maio de 2002

MINISTÉRIO DA CIÊNCIA E TECNOLOGIA
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

EXTENSÃO DO SGBD POSTGRESQL COM OPERADORES ESPACIAIS

Gilberto Ribeiro de Queiroz

Proposta de Mestrado em Computação Aplicada, orientada pelo Prof. Dr. João Argemiro C. Paiva e pelo Prof. Dr. Gilberto Câmara.

INPE
São José dos Campos
Maio de 2002

RESUMO

Este trabalho tem por objetivo a construção de uma extensão geográfica com operadqores espaciais para o SGBD PostgreSQL e sua integração com a biblioteca TerraLib. Para tanto, inicialmente são apresentados alguns conceitos básicos sobre Sistemas de Bancos de Dados, Sistemas de Informação Geográficas, Sistemas de Bancos de Dados Espaciais e as extensões espaciais comerciais existentes atualmente. A fim de determinar a eficiência da extensão e sua integração na biblioteca, é proposto o desenvolvimento de um aplicativo para testes.

SUMÁRIO

Pág.

LISTA DE FIGURAS
LISTA DE TABELAS

CAPÍTULO 1 –INTRODUÇÃO 1
1.1 – Objetivo 1
1.2 – Motivação 2
1.3 – Esboço Geral 2

CAPÍTULO 2 –SGBDS, GIS e SISTEMAS DE BANCOS DE DADOS ESPACIAIS 3
2.1 – SGBD-R 3
2.2 – SGBD-OR 4
2.3 – Arquiteturas de GIS 4
2.3.1 – Arquitetura Dual 5
2.3.2 – Arquitetura Integrada 6
2.4 – Sistemas de Bancos de Dados Espaciais 8
2.4.1 – Modelo de Dados 8
2.4.2 – Relacionamento Espacial 9
2.4.3 – Indexação Espacial 11
2.4.4 – Extensão da Linguagem de Consulta 14

CAPÍTULO 3 –OPENGIS e EXTENSÕES ESPACIAIS COMERCIAIS 17
3.1 – OpenGIS 17
3.1.1 – Modelo de Dados das Geometrias 17
3.1.2 – Operadores para Relacionamentos Topológicos 18
3.1.3 – Outros Operadores Importantes 19
3.1.4 – Arquitetura de Implementação das Tabelas com Feições 19
3.2 – Extensões Espaciais Comerciais 21
3.2.1 – IBM DB2 Spatial Extender 21
3.2.2 – Oracle Spatial 23

CAPÍTULO 4 –METODOLOGIA e RESULTADOS ESPERADOS 25

4.1 – PostgreSQL 25
4.2 – PostGIS 26
4.3 – Metodologia 28
4.3.1 – Definição dos Operadores Espaciais 28
4.3.2 – Implementação e Integração dos Operadores 28
4.3.3 – *Driver* de Bancos de Dados TerraLib 33
4.4 – Resultados Esperados 36
4.5 – Cronograma 37

REFERÊNCIAS BIBLIOGRÁFICAS 39

LISTA DE FIGURAS

Pág.

2.1 Exemplo de dado geográfico 5

2.2 Esquema da Arquitetura Dual 6

2.3 Esquema da Arquitetura Integrada 7

2.4 Relacionamentos Topológicos e Matriz de 4-Interseções 10

2.5 Relacionamentos Topológicos e Matriz de 9-Interseções 10

2.6 Cenário do Problema 11

2.7 Esquema da R-Tree 13

2.8 Esquema do Método de Indexação Grid-File 13

2.9 Esquema da QuadTree 14

3.1 Hierarquia de classes das geometrias 18

3.2 Esquema das tabelas da SFSSQL 20

3.3 Tipos de Dados Espaciais do Spatial Extender 22

4.1 Arquitetura da TerraLib para construção de bancos geográficos 35

LISTA DE TABELAS

Pág.

2.1 Esquema das Tabelas 14

3.1 Possíveis combinações de tipos geométricos 21

3.2 Esquema das Tabelas no Spatial Extender 22

3.3 Esquema das Tabelas no Oracle Spatial 24

4.1 Tabela de Metadados do Sistema de Coordenadas 27

4.2 Tabela de Metadados das Colunas com Geometria 27

4.3 Cronograma - Junho/2002 a Janeiro/2003 37

CAPÍTULO 1

INTRODUÇÃO

Atualmente, um dos maiores desafios de um *Sistema de Informação Geográfica* (GIS) é gerenciar de forma eficiente bancos de dados com informações bastante complexas através de um Sistema Gerenciador de Bancos de Dados (SGBD). Existem, basicamente, duas famílias de SGBDs: os Relacionais (SGBD-R) [6] e os Objeto-Relacionais (SGBD-OR) [30].

Os SGBD-Rs são fortemente voltados para aplicações corporativas (ou de automação comercial), manipulando grandes volumes de dados simples, isto é, dados que não levam em consideração, por exemplo, a componente espacial de um dado geográfico. Reconhece-se que tanto o modelo de dados quanto os operadores e métodos de acesso disponíveis nos SGBD-Rs não são suficientes para atender a todos os tipos de aplicações, [31], [27], [12], como no caso do GIS.

Este trabalho explora a nova geração de SGBDs Objeto-Relacionais extensíveis, com o intuito de contribuir para uma maior integração dessa tecnologia e a dos GIS através do desenvolvimento de uma extensão geográfica para um SGBD-OR.

1.1 Objetivo

O presente trabalho tem por objetivo desenvolver uma extensão geográfica com operadores espaciais, gratuita e de código fonte aberto, chamada **PostGIS** [25]. A definição dos operadores será baseada nas especificações do **OpenGIS** [23]. A extensão irá permitir ao SGBD PostgreSQL [32], também gratuito e de código fonte aberto, gerenciar informações geo-espaciais.

Como complemento, será desenvolvido um *driver* de banco de dados para a TerraLib [4] e [5], chamado **TePostGIS**, e um aplicativo, chamado **Spatial Database Explorer**, para servir como ambiente de testes do *driver* e da extensão. A finalidade do *driver* é realizar a integração entre a biblioteca e a extensão a ser desenvolvida, fornecendo meios para armazenar e recuperar dados geográficos, além de executar consultas espaciais fornecidas por esta extensão.

1.2 Motivação

A principal motivação é estar contribuindo para a disseminação do software gratuito e de código fonte aberto, pois não há disponível uma extensão geográfica deste tipo que possa servir como alternativa à produtos comerciais de custos elevados como o Oracle Spatial [26] e o IBM DB2 Spatial Extender [8]. Outra motivação é estar contribuindo para o desenvolvimento da biblioteca TerraLib, também gratuita e de código fonte aberto, através do desenvolvimento de um *driver* de bancos de dados. Esta biblioteca facilitará o desenvolvimento de aplicativos GIS, dando um grande incentivo à disseminação da tecnologia de Geoprocessamento.

1.3 Esboço Geral

Esta proposta de dissertação está dividida segundo os *seguintes* capítulos:

- O **Capítulo 2** faz uma breve revisão teórica sobre a tecnologia de Sistemas de Bancos de Dados tradicionais (SGBD-R e SGBD-OR), dos GIS e Sistemas de Bancos de Dados Espaciais apresentando seus modelos e arquiteturas;
- O **Capítulo 3** apresenta a proposta do OpenGIS para extensão da SQL e duas das principais extensões espaciais comerciais existentes atualmente;
- O **Capítulo 4** apresenta a metodologia a ser utilizada no desenvolvimento deste trabalho, bem como os resultados esperados e cronograma.

CAPÍTULO 2

SGBDS, GIS e SISTEMAS DE BANCOS DE DADOS ESPACIAIS

Além de ser citado em vários trabalhos, como [31], [27], [12] e [29], a prática vem demonstrando que os SGBDs podem desempenhar um papel fundamental no desenvolvimento de novos GIS, bastando para isso que eles estejam "preparados"para esta nova missão. Assim como eles desempenham um papel fundamental nas aplicações corporativas, liberando os programadores de aplicação para se preocuparem em resolver os problemas do domínio do negócio em si e não no desenvolvimento de funcionalidades básicas de armazenamento, tais como recuperação, controle de integridade e concorrência sobre os dados, eles podem beneficiar os desenvolvedores de GIS. Por exemplo, permitindo que estes enfoquem nas funcionalidades que um GIS deve ter, como ferramentas para análise espacial, visualização gráfica dos dados geográficos e entrada de dados, ao invés de se preocuparem com o gerenciamento desses dados. Outro ponto importante é que o armazenamento de dados dentro de um SGBD elimina o problema de uso de estruturas proprietárias que dificultam a interoperabilidade de dados.

2.1 SGBD-R

O modelo de dados dos SGBD-R consiste de uma coleção de relações, cada qual com atributos de um tipo específico (domínio). Nos sistemas comerciais de bancos de dados atuais esses tipos incluem números inteiros, de ponto flutuante, cadeias de caracteres, datas e campos binários longos (BLOBs). Para esses tipos encontram-se disponíveis uma variedade de operadores (com exceção ao BLOB), como operações aritméticas, de conversão, de manipulação textual e operações com data.

Outra característica dos SGBD-R é que eles fornecem eficientes mecanismos de armazenamento, acesso (indexação de dados unidimensionais), recuperação, controle de concorrência e integridade dos dados armazenados. No entanto, é reconhecido que este modelo não é suficiente para cobrir todos os tipos de aplicações. Por exemplo, as componentes espaciais de um dado geográfico devem utilizar mecanismos de indexação especiais (multidimensionais), como a R-Tree [17].

Stonebraker (1996) aponta que a simulação de tipos de dados (através de um esquema de tabelas), como os geográficos, em SGBD-R podem ter efeitos colaterais como queda de desempenho e dificuldade de codificação e posterior manutenção de código

da aplicação. Ele cita um exemplo clássico de aplicação, um GIS, que ao tentar simular os tipos de dados apresenta uma série de problemas. A fim de realizar uma maior integração entre aplicações não-tradicionais (caso dos GIS) e os SGBDs, é necessário utilizar uma outra tecnologia de bancos de dados emergente, os SGBD-OR.

2.2 SGBD-OR

Os SGBD-OR estendem o modelo relacional, e entre suas principais características está a capacidade de fornecer um sistema de tipos muito mais rico, passível de extensão e que possa ser utilizado em linguagens de consulta como a SQL. Além da definição de novos tipos, há a possibilidade de definição de funções e operadores para manipulação desses tipos. Outro aspecto interessante diz respeito à possibilidade de definição de novos mecanismos de indexação sobre tipos definidos pelo usuário. Isso faz dos SGBD-OR uma boa solução na integração com os GIS.

A possibilidade de extender os tipos de dados em um SGBD-OR elimina os problemas de eficiência ocorridos na simulação de tipos, uma vez que o usuário pode definir os tipos de dados, as operações e métodos de acesso, dependendo do tipo de aplicação que ele queira construir. Outro importante fator, é que só armazenar os dados no banco não facilita, na prática, a interoperabilidade de dados, uma vez que ao utilizar simulação de tipos, cada categoria de aplicação pode desenvolver seu próprio modelo para construir as geometrias básicas. Nesse caso, a aplicação passa a trabalhar sobre dois modelos conceituais diferentes, um para a geometria e outra para a aplicação. Portanto, o programador da aplicação precisa desenvolver o esquema das geometrias e o código para explorar relacionamentos espaciais. Na definição de novos tipos, os tipos geométricos básicos já estão definidos, sendo necessário apenas desenvolver o modelo de dados da aplicação. Dessa forma, a representação dos tipos geométricos básicos será sempre a mesma.

2.3 Arquiteturas de GIS

Os dados que um Sistema de Informação Geográfica (GIS) manipulam representam objetos e fenômenos, cuja localização geográfica é uma das principais características utilizadas em sua análise. Esses dados possuem duas componentes principais:

- **Atributos não espaciais:** também chamados de atributos alfanuméricos, que são um conjunto de atributos usados para descrever o dado geográfico de estudo, como por exemplo, nome e população de um

país (Tabela na Figura 2.1);

- **Atributo espacial ou componente espacial:** especifica a localização geográfica e a forma (geometria) do objeto em estudo. Por exemplo, um país pode ser representado por um polígono no espaço bi-dimensional (Mapa da Figura 2.1).

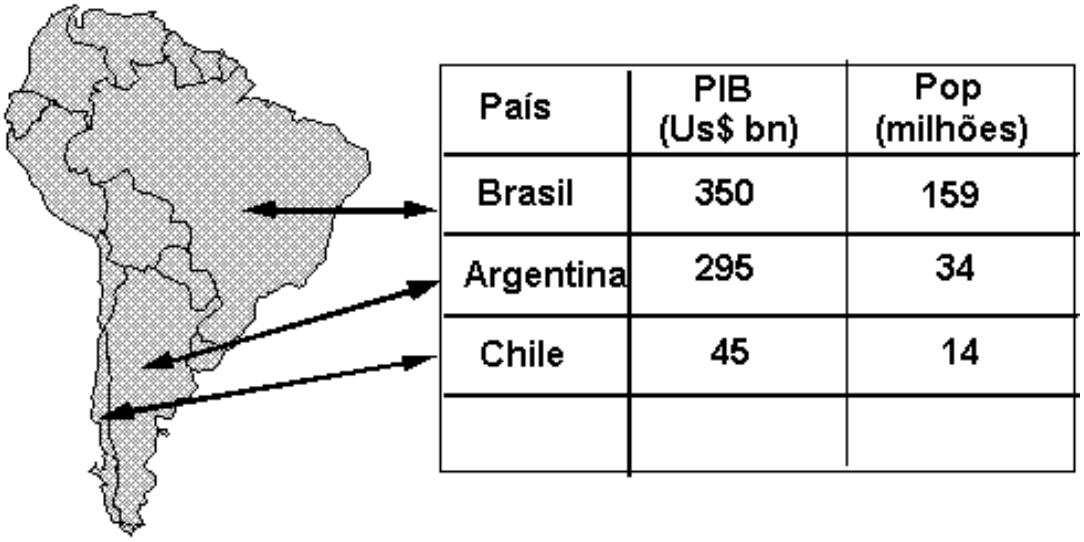

Fig. 2.1 – Exemplo de dado geográfico

Existem, basicamente, duas arquiteturas para os GIS, no que diz respeito à sua integração com os SGBDs: a arquitetura Dual e a Integrada [28]. Elas diferem-se, principalmente, na maneira e nos recursos utilizados para armazenar e recuperar dados geográficos.

2.3.1 Arquitetura Dual

A arquitetura Dual caracteriza-se por utilizar um sistema de banco de dados relacional (SGBD-R), no gerenciamento da parte não espacial dos dados geográficos e formatos proprietários para armazenar os atributos espaciais, usando um identificador comum na interligação dos dois tipos de atributos. A Figura 2.2 ilustra esta arquitetura.

As vantagens que esse sistema traz diz em respeito, principalmente, à atribuição de parte do gerenciamento dos dados aos SGBDs, pois os sofisticados mecanismos

de transação, recuperação contra falhas, concorrência, integridade dos dados e indexação podem ser usados para a parte não espacial dos dados. No entanto, esta arquitetura apresenta a falha de não integrar totalmente o dado geográfico, o que torna necessário para a aplicação desenvolver mecanismos que façam o controle de integridade, com a finalidade de evitar problemas de inconsistência, e também mecanismos para a indexação da parte espacial. Outro problema apresentado diz respeito às consultas diretamente em SQL que ficam restritas à parte alfanumérica. O **SPRING** [3] é um exemplo de sistema que utiliza este tipo de arquitetura.

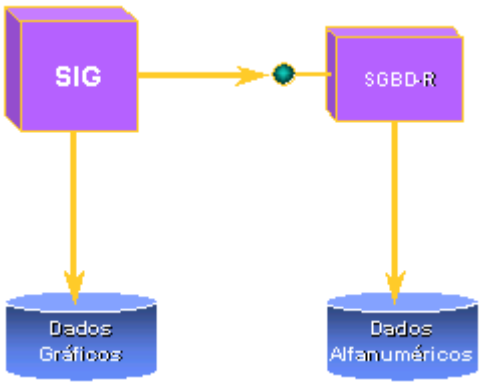

Fig. 2.2 – Esquema da Arquitetura Dual
Fonte: adaptado de [7]

2.3.2 Arquitetura Integrada

Esta arquitetura caracteriza-se por armazenar no banco de dados tanto os atributos espaciais quanto os alfanuméricos. São utilizadas, basicamente, três estratégias na implementação desta arquitetura: a utilização de campos binários longos (BLOBs), definição de um esquema de tabelas para os atributos espaciais, ou a utilização de extensões espaciais de bancos de dados objetos-relacionais. A Figura 2.3 ilustra esta arquitetura.

Na primeira estratégia, os atributos espaciais são armazenados em BLOBs disponíveis na maioria dos SGBDs, inclusive os relacionais. A vantagem desse tipo de solução é a integração dos dados em um único ambiente, com a possibilidade de utilização dos mecanismos de controle de integridade, transação e controle de concorrência, eliminando possíveis problemas de inconsistência. Por outro lado, essa idéia esbarra no problema do pouco conhecimento que o SGBD possuí sobre o dado

armazenado como BLOB (na verdade o BLOB é considerado uma "caixa preta"). O SGBD não fornece mecanismo para possibilitar a criação de índices ou criação de operações sobre estes campos.

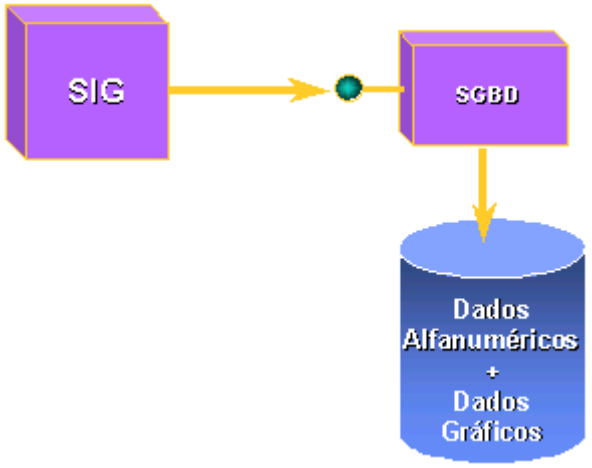

Fig. 2.3 – Esquema da Arquitetura Integrada
Fonte: adaptado de [7]

A segunda estratégia possibilita o desenvolvimento de recursos de indexação espacial, mas restritos ao mapeamento para uma estrutura unidimensional, como uma B-Tree [14]. Uma forma de resolver estes problemas é utilizar a terceira estratégia (através de extensões). Alguns SGBDs já dispõem de extensões espaciais, como o Oracle e o IBM DB2, e outros possuem mecanismos de extensibilidade, que permitem a definição de novos tipos de dados e a criação de operações sobre estes novos tipos (PostgreSQL, por exemplo). Entre as vantagens da adoção desta classe de banco de dados, pode-se citar:

- Existência de excelentes mecanismos de indexação, tanto de atributos convencionais quanto de atributos espaciais;
- Funções já definidas para tratamento dos atributos espaciais (operadores topológicos e métricos, por exemplo).

Uma das desvantagens desse tipo de solução é a falta de padronização das linguagens de consultas (operadores e tipos de dados). Com tudo, existe um esforço do consórcio OpenGIS, apoiado pelas principais líderes do mercado mundial de bancos de dados

geográficos, como Oracle, IBM, Informix, ESRI e MapInfo, para criar um esquema padronizado da SQL que dê suporte aos dados geográficos. Este esquema encontra-se em uma especificação conhecida como Simple Feature Specification For SQL ou, abreviadamente, **SFSSQL** [23] e trata-se da especificação na qual esta proposta se baseia. O Capítulo 3 apresenta os principais tópicos desta especificação relevantes para este projeto.

2.4 Sistemas de Bancos de Dados Espaciais

Muitas pesquisas têm sido feitas na área de sistemas de bancos de dados espaciais (SBDE), entre elas, pode-se citar [29], [21] e [16], que apontam os principais requisitos, técnicas e linhas de pesquisa. Os requisitos e funcionalidades de um SBDE são bastante influenciados pelos GIS, pois estes são a classe de aplicação que mais impulsionam as pesquisas nesta área. Este trabalho considera os SBDE como sendo uma tecnologia de suporte ao desenvolvimento de GIS, de forma a fornecer apenas algumas funcionalidades básicas de gerenciamento de dados. Portanto, não é considerado o uso de um SBDE diretamente como um SIG. Por exemplo, as funcionalidades de visualização e entrada dos dados são consideradas de responsabilidade de implementação do SIG e não do SGBD. Nas subseções seguintes são apresentadas as características básicas de um SBDE.

2.4.1 Modelo de Dados

Câmara et al. (1996) discute que há duas maneiras de encarar a realidade: a visão de campos e a visão de objetos. O termo SBDE está fortemente associado à segunda visão. Dessa forma, as técnicas utilizadas são as de tratamento de objetos com identidade, limites e relacionamentos bem definidos (representação vetorial). Para poder descrever os tipos de objetos espaciais, deve-se introduzir um modelo de dados espacial, uma abstração que esconde os detalhes de armazenamento físico do dado. Nesse contexto, um SBDE deve dar suporte a tipos de dados espaciais (SDTs), como ponto, linha e polígono, e oferecer recursos para que estes dados sejam manipulados, assim como os tipos alfanuméricos básicos (inteiros, strings e datas). Para isso, sua linguagem de consulta deve ser estendida, permitindo manipular SDTs e suportando operadores espaciais que tratem das relações espaciais entre eles.

2.4.2 Relacionamento Espacial

Conforme Egenhofer (1991), o entendimento formal dos relacionamentos espaciais entre objetos é de fundamental importância para a análise de dados nos GIS, pois estes relacionamentos são, freqüentemente, envolvidos em consultas espaciais entre os objetos. Portanto, é crucial para o projeto de um sistema de bancos de dados espaciais adotar uma ferramenta que permita descrever o relacionamento entre os objetos.

Existem vários tipos de relacionamentos espaciais, mas os mais importantes para um SBDE são: relacionamentos métricos, relacionamentos direcionais e relacionamentos topológicos. No caso dos relacionamentos topológicos, existem várias propostas de modelos com o objetivo de descrever os relacionamentos possíveis entre dois objetos. Entre eles, pode-se citar a Matriz de 4-Interseções [10], Matriz de 9-Interseções [11] e a Matriz de 9-Interseções Dimensionalmente Extendida [23].

Em Egenhofer (1995) são apresentadas oito relações topológicas entre dois objetos, baseadas nas interseções de interiores (O°) e fronteiras (∂O). Esse modelo é conhecido como 4-Interseções, podendo ser representado em uma matriz 2x2. A Figura 2.4 apresenta os oito relacionamentos e a respectiva configuração da matriz de interseções.

O modelo de 9-Interseções é uma extensão do 4-Interseções, acrescentando a comparação entre os exteriores (O^-). Isto é feito a fim de distinguir algumas configurações topológicas diferentes, que não são detectadas no primeiro modelo citado e para detectar relacionamentos entre regiões com ilhas. A Figura 2.5 ilustra este modelo.

O modelo de 9-Interseções Dimensionalmente Extendido inclui a informação da dimensão dos resultados dos testes de interseção. Este modelo permite expressar o relacionamento entre pontos, linhas e regiões incluindo regiões com ilhas e linhas e regiões compostas.

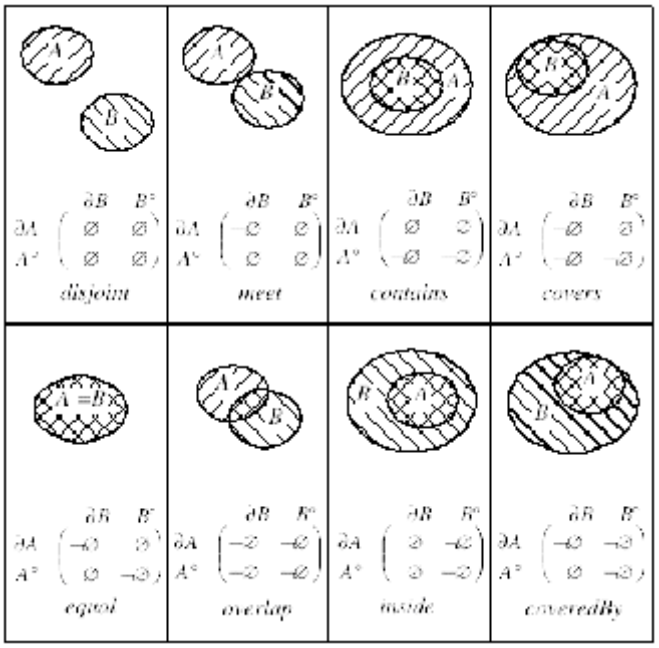

Fig. 2.4 – Relacionamentos Topológicos e Matriz de 4-Interseções
Fonte: [9]

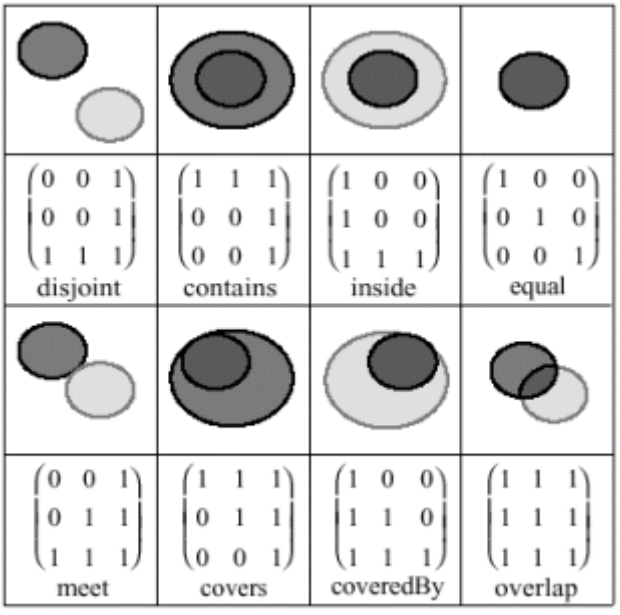

Fig. 2.5 – Relacionamentos Topológicos e Matriz de 9-Interseções

Um cenário, prático, que ilustra uma consulta empregando relacionamentos espaciais é: "O oficial encarregado do serviço contra incêndios precisa de uma lista de todas as áreas sensíveis dentro de um raio de 8 km de uma área de lixo tóxico". A Figura 2.6 representa o esquema dessa consulta.

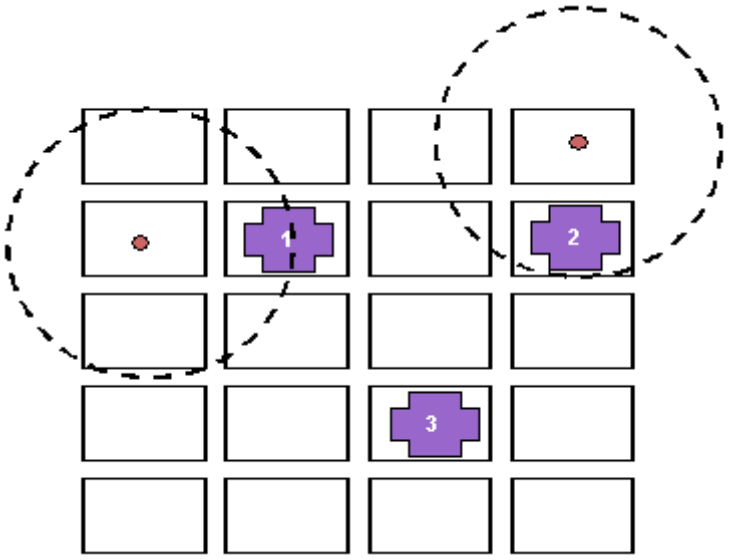

Fig. 2.6 – Cenário do Problema

Para resolver a consulta acima, é necessário, por exemplo, a definição de operadores espaciais para gerar um mapa de distância (*buffer*) e para testar a interseção. O primeiro geraria uma região de influência (de 8 Km) a partir dos pontos de lixo tóxico e o segundo informaria se uma área está ou não dentro do raio de influência das áreas de lixo tóxico. Além dos relacionamentos espaciais, outro ponto muito importante para um SBDE é a utilização de técnicas que otimizem as consultas espaciais.

2.4.3 Indexação Espacial

Um método de indexação espacial é uma técnica de otimização de acesso aos dados, a fim de acelerar as consultas espaciais. Isto é feito, geralmente, reduzindo o conjunto de dados (espaço de busca) a ser processado. Essas técnicas normalmente empregam a estratégia de processamento de uma consulta em dois passos:

- No primeiro passo, o de filtragem, é feito o uso de aproximações da geo-

metria exata, pois os relacionamentos espaciais envolvem algoritmos geométricos computacionais complexos e custosos. A técnica mais comum de aproximação é o uso do menor retângulo envolvente (MBR);

- No segundo passo, o de refinamento, a geometria exata é comparada para os objetos selecionados no primeiro passo.

Exemplos de métodos de indexação são:

- **R-Tree** [2]: o método de indexação R-Tree pertence à categoria dos métodos de acesso a dados ordenados. Assim como a B-Tree, é baseado em uma estrutura hierárquica balanceada (árvore balanceada), onde cada nó, se interno ou folha, é mapeado para uma página no disco. A R-Tree é construída sobre retângulos distribuídos no plano a ser indexado e cada nó está associado a um desses retângulos. Na árvore R-Tree, os nós não folhas contém entradas somente para as subárvores e apenas os nós folhas terão entradas para os objetos do espaço indexado. Para acessar um retângulo da coleção indexada, percorre-se um caminho da raiz da árvore até um dos vários nós folhas, testando se o retângulo de cada nível contém ou sobrepõe o retângulo desejado. Sendo assim, as consultas são mais lentas, pois diversos caminhos precisam ser pesquisados. Seu esquema é apresentado na Figura 2.7.

- **Grid-File** [27]: neste método de indexação o espaço é particionado em uma grade retangular, onde cada célula é associada a uma página. Uma geometria G é associada às páginas das células Ci da grade que interceptam com o retângulo envolvente da geometria. Neste tipo de índice é criado uma matriz bi-dimensional, conhecido como diretório, onde os elementos possuem o endereço de uma página. A Figura 2.8 ilustra a representação deste tipo de indexação. Conforme pode ser observado, o retângulo envolvente de uma certa geometria pode sobrepor mais de uma célula. Caso isso aconteça com muita freqüência a eficiência do índice pode sofrer uma degradação. Essa característica torna a definição da resolução da grade muito importante no projeto deste tipo de índice. No entanto, pode-se optar por realizar vários níveis de grade.

- **QuadTree** [27]: este método particiona o espaco em quadrantes, e sua ilustração encontra-se na Figura 2.9.

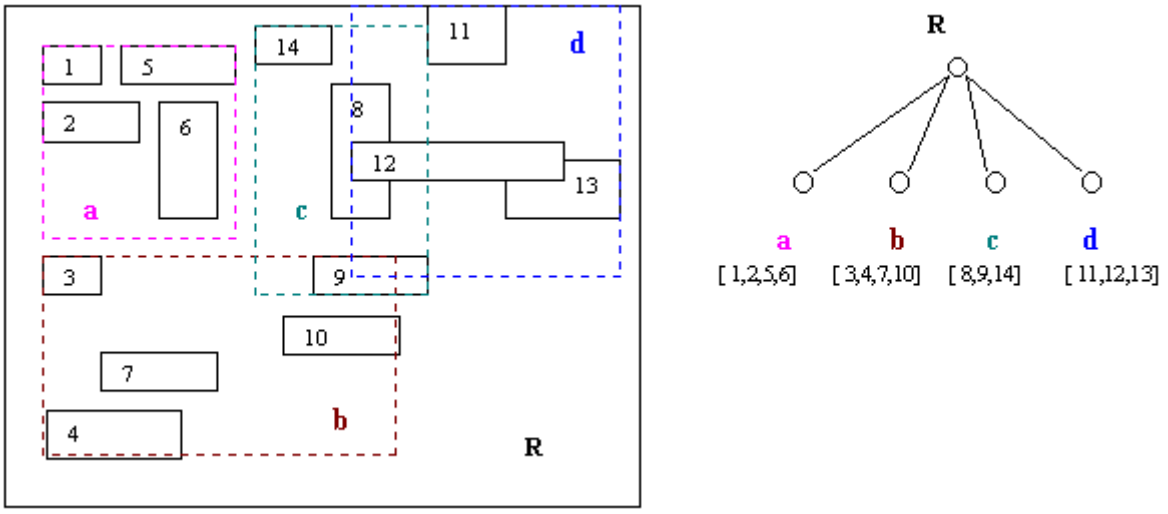

Fig. 2.7 – Esquema da R-Tree

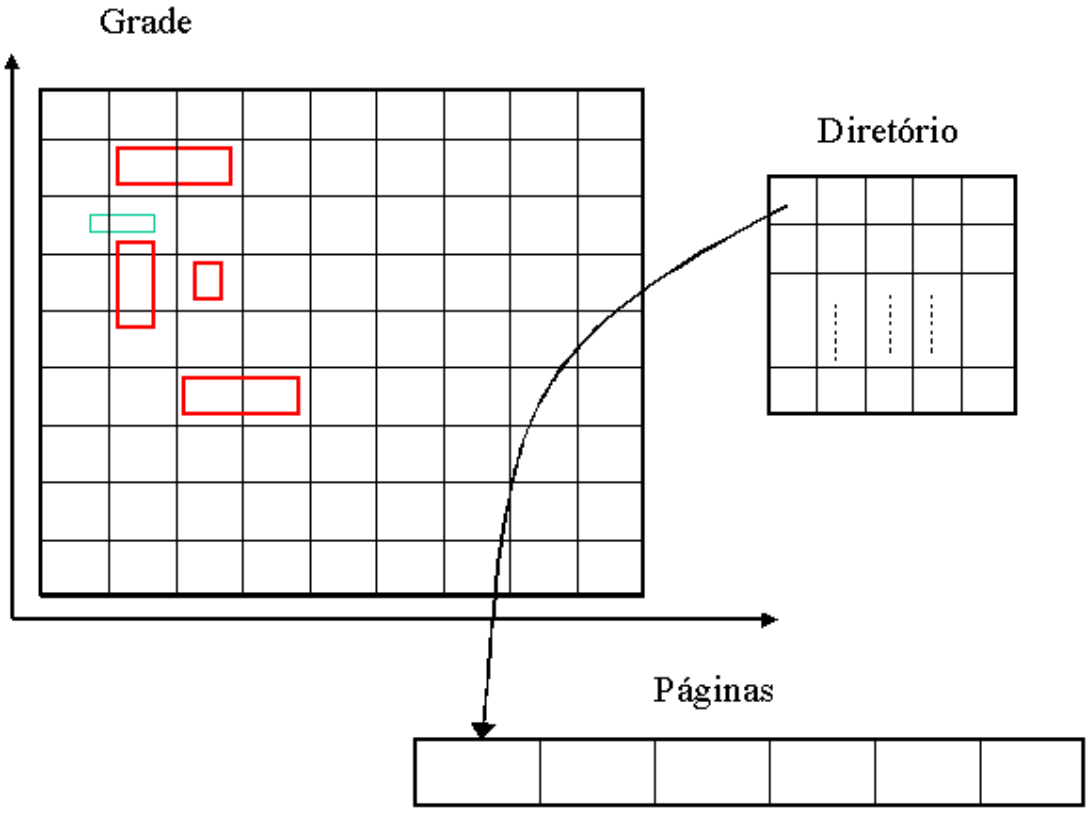

Fig. 2.8 – Esquema do Método de Indexação Grid-File

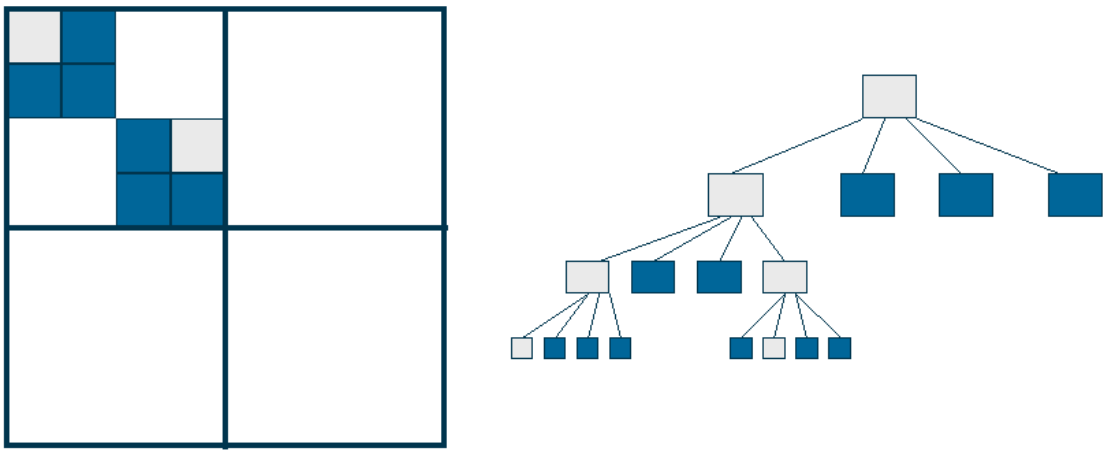

Fig. 2.9 – Esquema da QuadTree

2.4.4 Extensão da Linguagem de Consulta

A linguagem de consulta mais empregada atualmente é a SQL. No entanto, ela não prevê em sua definição, tipos de dados e operadores espaciais. Um SBDE deve embutir na SQL os tipos de dados espaciais, para que os seus usuários possam tratá-los como tipos normais (inteiro, caracter e data). Por exemplo, um SBDE deveria dar suporte à definição de tabelas com atributos espaciais e à construção de uma consulta, ambos em SQL, para satisfazer o problema da Seção 2.4.2 da seguinte forma:

- Definição das tabelas com os tipos geométricos:

zona_sensivel	
Nome Atributo	**Tipo**
nome	STRING
zona	POLÍGONO

area_risco	
Nome Atributo	**Tipo**
nome	STRING
local	PONTO

TABELA 2.1 – Esquema das Tabelas

- Consulta em SQL utilizando operadores espaciais:

```
SELECT as.name, ar.name FROM AREAS_SENSIVEL as,
                             AREAS_RISCO ar
 WHERE Intercepta(Buffer(ar.local, 8000),as.zona)
```

Conforme a Seção 2.3.2, a falta de padronização na adição de tipos e operadores espaciais na SQL pode levar a dialetos diferentes de SQL. No entanto, há um esforço da comunidade de desenvolvimento de extensões geográficas em se ter um consenso na incorporação dos tipos e operadores na SQL. Esse consenso encontra-se na especificação conhecida como Simple Feature Specification For SQL (SFSSQL), cujos principais pontos serão apresentados no Capítulo 3.

CAPÍTULO 3

OPENGIS e EXTENSÕES ESPACIAIS COMERCIAIS

Este capítulo apresenta a proposta do OpenGIS para extensão da SQL e as duas principais extensões espaciais comerciais existentes atualmente, o Oracle Spatial e o IBM DB2 Spatial Extender.

3.1 OpenGIS

A proposta do OpenGIS (OGIS) consiste na especificação de um esquema SQL padrão que permita o armazenamento, recuperação, consulta e atualização de coleções de feições geo-espaciais (dados geográficos). Esta especificação está limitada às operações topológicas e métricas em tipos de dados vetoriais. Ela introduz o conceito de "tabela com feições"para representação dos dados geográficos. Nesta tabela, os atributos não espaciais são mapeados para colunas de tipos disponíveis na SQL92, e a componente espacial para colunas cujo tipo de dados é baseado no conceito de "tipos de dados geométricos adicionais para SQL".

As colunas geométricas podem seguir dois modelos, o SQL92 e o SQL92 com Tipos Geométricos. O primeiro consiste na utilização de uma tabela para representar as colunas espaciais. A segunda, utiliza o conceito de tipos abstratos de dados, sendo a coluna representada por um tipo geométrico que extende os tipos da SQL. Como este trabalho consiste na proposta de uma extensão com tipos de dados espaciais que estendam a SQL, apenas os detalhes do segundo modelo serão apresentados.

3.1.1 Modelo de Dados das Geometrias

A Figura 3.1 ilustra a hierarquia de tipos definida pela especificação do OGIS. Este diagrama é o mesmo tanto para o modelo da SQL92 quanto para o da SQL92 com Tipos Geométricos.

O modelo acima representa objetos com 0, 1 ou 2 dimensões no espaço bidimensional. Algumas classes são abstratas como: Curve, Surface, MultiSurface e MultiCurve. Uma classe especial é a GeometryCollection, que pode ser composta por mais de um tipo de geometria (tipo heterogêneo). As outras classes, são tipos básicos, como no caso de Point, LineString e Polygon, que podem formar coleções homogêneas como MultiPoint, MultiLineString e MultiPolygon, respectivamente. Cada uma dessas

classes possui uma série de atributos, métodos e definições que são apresentadas na especificação. No caso deste trabalho, a ênfase maior é nos métodos para determinação dos relacionamentos espaciais entre os objetos, que são apresentados na seção seguinte.

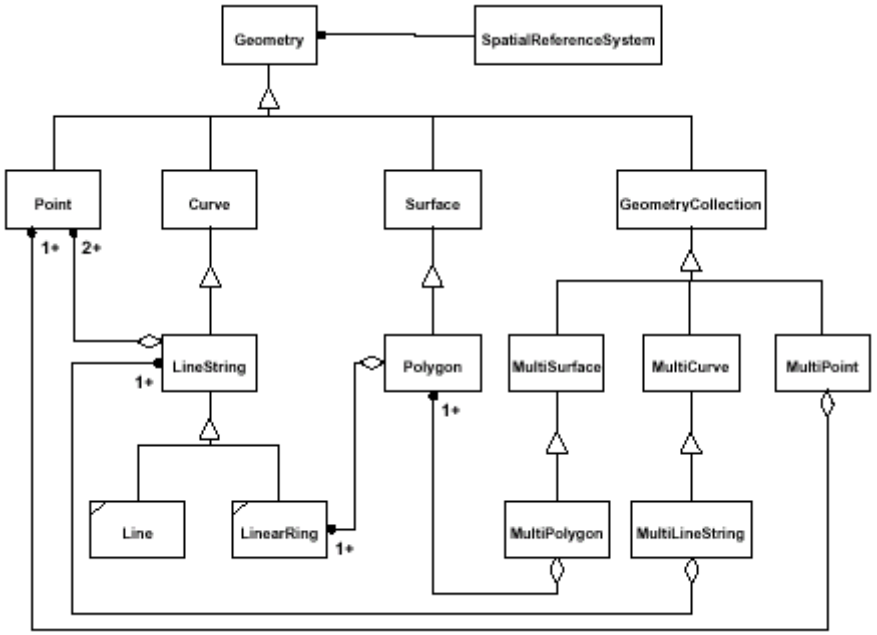

Fig. 3.1 – Hierarquia de classes das geometrias
Fonte: [23]

3.1.2 Operadores para Relacionamentos Topológicos

Os operadores de relacionamento topológico são definidos como métodos da classe Geometry e, portanto, servem para testar os relacionamentos topológicos entre quaisquer objetos do modelo anterior. Os seguintes operadores estão definidos: Equals, Disjoint, Intersects, Touches, Crosses, Within, Contains, Overlaps. Todos são métodos unários que recebem uma outra geometria como parâmetro de entrada e retornam o valor inteiro 0 (FALSE) se o relacionamento não for satisfeito e 1 (TRUE) caso este seja satisfeito.

Além desses oito operadores, um outro operador, chamado Relate, é definido, recebendo a geometria a ser comparada e um segundo parâmetro que representa um padrão da matriz de interseção na forma de uma string de nove caracteres (teste en-

tre interseção de fronteira, interior e exterior). Ele retorna o inteiro 1 (TRUE) se as geometrias forem relacionadas espacialmente de acordo com os valores especificados na matriz. O OGIS utiliza o modelo de 9-Interseções Dimensionalmente Extendido [23].

3.1.3 Outros Operadores Importantes

Além dos operadores topológicos, a especificação define vários outros métodos para a classe Geometry que são comumente empregados pelos GIS. Entre eles podemos citar:

- Distance(outraGeometria:Geometry):Double = retorna a distância entre as geometrias;
- Buffer(distância:Double):Geometry = retorna uma geometria definida por mapa de distância;
- ConvexHull():Geometry = retorna um polígono convexo que contém todos os pontos da geometria;
- Intersection(outraGeometria:Geometry):Geometry = retorna a geometria resultante da interseção das geometrias;
- Union(outraGeometria:Geometry):Geometry = retorna a geometria resultante da união de duas geometrias;
- Difference(outraGeometria:Geometry):Geometry : retorna a geometria resultante da diferença entre as geometrias.

A classe Surface e MultiSurface ainda apresentam especificamente:

- Area():Double = área de uma região;
- Centroid():Point = um ponto representando o centróide da geometria;
- PointOnSurface():Point = um ponto que esteja na superfície.

3.1.4 Arquitetura de Implementação das Tabelas com Feições

O OGIS propõe o seguinte esquema (Figura 3.2) para representação dos dados geográficos para tabelas com atributos de Tipos Geométricos na SQL:

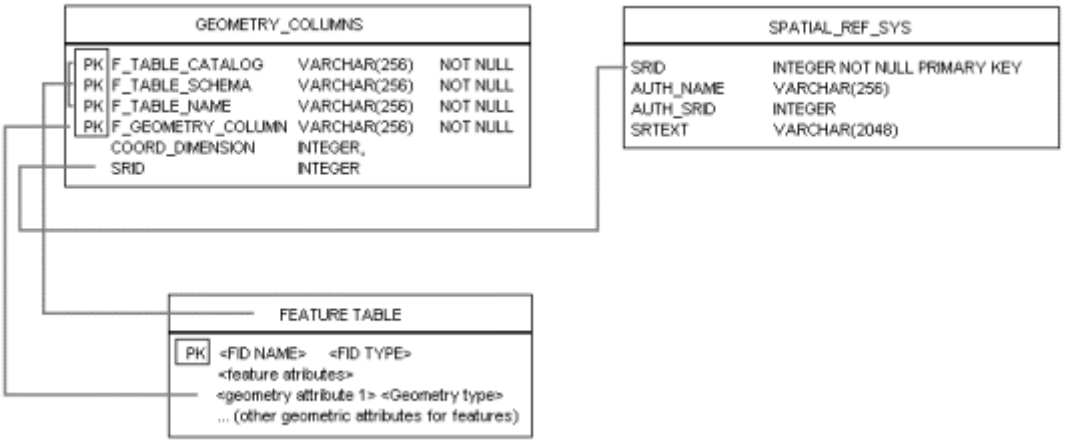

Fig. 3.2 – Esquema das tabelas da SFSSQL
Fonte: adaptado de [23]

A tabela SPATIAL_REF_SYS armazena as informações de cada sistema de coordenadas (SRS) utilizado no banco de dados. A tabela GEOMETRY_COLUMNS serve como tabela de metadado para as colunas geométricas das tabelas com feições. Nestas tabelas, cada instância de uma geometria deve ser associada com um SRS de forma a permitir associações de SRS a instâncias que ainda não estejam armazenadas no banco. Isso é útil para que as funções possam verificar a compatibilidade dos SRS de objetos. Além das tabelas, a especificação cita que a SQL deve suportar um subconjunto dos tipos mostrados no diagrama da Figura 3.1. A Tabela 3.1 mostra os possíveis tipos que uma implementação deva suportar.

Uma coluna pode ser de qualquer um dos tipos disponíveis. Uma coluna declarada como sendo de um certo tipo pode armazenar quaisquer instâncias deste tipo além de instâncias de seus subtipos. A seguir, serão apresentadas duas extensões espaciais comerciais desenvolvidas com base nesse esquema.

Nível	Tipos Disponíveis	Tipos Instanciáveis
1	Geometry, Point, Curve, LineString, Surface, Polygon, GeomCollection	Point, LineString, Polygon, GeometryCollection
2	Geometry, Point, Curve, LineString, Surface, Polygon, GeomCollection, MultiPoint, MultiCurve, MultiLineString, MultiSurface, MultiPolygon	Point, LineString, Polygon, MultiPoint, MultiLineString, MultiPolygon
3	Geometry, Point, Curve, LineString, Surface, Polygon, GeomCollection, MultiPoint, MultiCurve, MultiLineString, MultiSurface, MultiPolygon	Point, LineString, Polygon, GeomCollection, MultiPoint, MultiLineString, MultiPolygon

TABELA 3.1 – Possíveis combinações de tipos geométricos

3.2 Extensões Espaciais Comerciais

Atualmente, existem basicamente duas extensões comerciais disponíveis no mercado para tratar de dados geográficos no formato vetorial: Oracle Spatial [22] e IBM DB2 Spatial Extender [19]. Essas extensões baseiam-se nas especificações do OpenGIS, porém, apresentam variações relevantes entre os modelos de dados, semântica dos operadores espaciais, mecanismos de indexação e esquema e sintaxe da SQL extendida com tipos espaciais. A seguir, são apresentadas as características e funcionalidades das extensões da IBM e Oracle.

3.2.1 IBM DB2 Spatial Extender

O Spatial Extender é uma extensão do DB2 UDB da IBM, desenvolvido pela ESRI e IBM para integrar funcionalidades espaciais ao DB2. Esta extensão procura seguir os padrões definidos na SFSSQL. A hierarquia de tipos em seu modelo de dados espacial pode ser visto na Figura 3.3:

Nesse esquema, uma geometria pode ser formada por um único elemento (ST_POINT, ST_LINESTRING, ST_POLYGON) ou por um conjunto homogêneo (ST_MULTIPOINT, ST_MULTILINESSTRING, ST_MULTIPOLYGON). Esses tipos são conhecidos como primitivos, sendo possível definir colunas do tipo ST_GEOMETRY que podem assumir qualquer um dos outros tipos. Uma tabela espacial pode ser formada por atributos alfanuméricos definidos como colunas de tipos básicos (VARCHAR, NUMBER) e por uma coluna do tipo espacial. Essa extensão fornece um conjunto de operadores e funções espaciais, que são utilizados juntamente

com a linguagem SQL, para suportar consultas espaciais. Os operadores topológicos seguem o Modelo de 9-Interseções Dimensionalmente Extendido. O mecanismo de indexação disponível é o Grid-File.

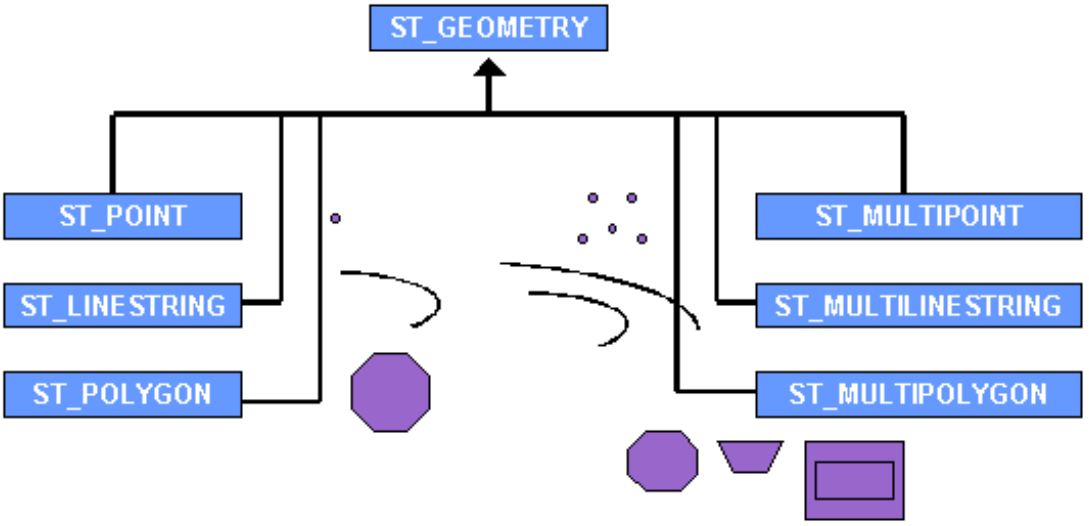

Fig. 3.3 – Tipos de Dados Espaciais do Spatial Extender

O exemplo da Seção 2.4.4 poderia ser traduzido da seguinte maneira:

- Definição de duas tabelas com atributos espaciais:

zona_sensivel	
Nome Atributo	**Tipo**
nome	VARCHAR(50)
zona	ST_POLYGON

area_risco	
Nome Atributo	**Tipo**
nome	VARCHAR(50)
local	ST_POINT

TABELA 3.2 – Esquema das Tabelas no Spatial Extender

- A sintaxe da SQL do Spatial Explorer seria:

```
SELECT as.name, ar.name FROM AREAS_SENSIVEL as,
                             AREAS_RISCO ar
 WHERE ST_Intersects(ST_Buffer(ar.local, 8000),as.zona)
```

3.2.2 Oracle Spatial

O Oracle Spatial é uma extensão espacial do SGBD Oracle, que utiliza seu modelo objeto-relacional. Esta extensão contém um conjunto de funcionalidades e procedimentos que permite armazenar, acessar e analisar dados espaciais em um banco de dados Oracle. Seu modelo de dados consiste em uma estrutura hierárquica de elementos, geometrias e layers; onde layers são compostos por geometrias, que por sua vez são compostas por elementos. Os elementos podem ser do tipo Point, LineString ou Polygon (com ou sem ilhas). Uma geometria pode ser formada por um único elemento ou por um conjunto homogêneo (MultiPoint, MultiLinesString ou MultiPolygon) ou heterogêneo (Collection) de elementos. E, finalmente, um layer é formado por um conjunto de geometrias que possuem os mesmos atributos. Devido à utilização de um modelo objeto-relacional, cada geometria é armazenada em um objeto chamado SDO_GEOMETRY, cuja estrutura é apresentada abaixo:

```
SDO_GEOMETRY
{
 SDO_GTYPE        NUMBER
 SDO_SRID         NUMBER
 SDO_POINT        SDO_POINT_TYPE /* Utilizado apenas se      */
                                 /* a geometria for um ponto */
 SDO_ELEM_INFO    SDO_ELEM_INFO_ARRAY
 SDO_ORDINATES    SDO_ORDINATE_ARRAY
}
```

Este objeto contém a geometria em si, suas coordenadas (SDO_ORDINATES), e informações sobre seu tipo (SDO_GTYPE) e projeção (SDO_SRID). Em uma tabela espacial, os atributos alfanuméricos da geometria são definidos como colunas de tipos básicos (VARCHAR2, NUMBER, etc) e a geometria, como uma coluna do tipo SDO_GEOMETRY. Sendo assim, cada tabela espacial armazena um layer, o qual é composto pelo conjunto de todas geometrias desta tabela. O Oracle Spatial fornece um conjunto de operadores e funções espaciais, que são utilizados juntamente com a linguagem SQL, para suportar consultas espaciais. Para consultar relações topológicas entre duas geometrias é utilizado um operador chamado SDO_RELATE. Este operador implementa o Modelo de 9-Interseções, considerando as interseções, vazia (0) ou não vazia (1), entre os interiores, fronteiras e exteriores de duas geometrias. O SDO_RELATE recebe como parâmetro o tipo de relação topológica que

deve ser computada. Os possíveis parâmetros são: Equal, Disjoint, Touch, Inside, OverlapBdyIntersect, OverlapBdyDisjoint, Anyinteract, Contains, On, Covers e Coveredby. Quanto à indexação espacial, esta extensão fornece dois tipos de índices R-tree e Quadtree. Cada um desses índices é apropriado para diferentes situações e podem ser usados simultaneamente para indexar uma mesma coluna geométria. Como exemplo, as tabelas e consulta mostradas na seção anterior utilizando os tipos, operadores e funções do Oracle Spatial seria:

zona_sensivel	
Nome Atributo	**Tipo**
nome	VARCHAR2(50)
zona	SDO_GEOMETRY

area_risco	
Nome Atributo	**Tipo**
nome	VARCHAR2(50)
local	SDO_GEOMETRY

TABELA 3.3 – Esquema das Tabelas no Oracle Spatial

```
SELECT ass.nome, ar.nome
  FROM AREA_RISCO ar, AREA_SENSIVEL ass,
       USER_SDO_GEOM_METADATA m
 WHERE m.table_name = 'AREA_RISCO' AND
      (SDO_RELATE(ass.zona,
       SDO_GEOM.SDO_BUFFER(ar.local, m.diminfo,
    8000), 'mask=ANYINTERACT querytype=WINDOW') = 'TRUE');
```

CAPÍTULO 4

METODOLOGIA e RESULTADOS ESPERADOS

Este trabalho propõe o desenvolvimento de operadores espaciais a serem incorporados na extensão geográfica PostGIS baseada no banco PostgreSQL, criação de um diver de banco de dados para esta extensão e criação de uma aplicação para testes. Para cumprir estes objetivos, a seguinte metodologia será utilizada:

- Definição dos Operadores Espaciais a serem implementados;
- Implementação e Integração dos Operadores ao PostGIS;
- Implementação do *driver* de Banco de Dados na TerraLib;
- Construção do aplicativo Spatial Database Explorer para realizações de testes com drivers de SGBDs da TerraLib.

4.1 PostgreSQL

O PostgreSQL é um sistema gerenciador de banco de dados objeto-relacional, gratuito e de código fonte aberto que desenvolveu-se a partir de um projeto chamado Postgres, iniciado por volta de 1986, na Universidade da Califórnia (Berkeley), sob a liderança do professor Michael Stonebraker. Em 1995 foi adicionado suporte a SQL e o código fonte, disponibilizado na Web. Desde então, um grupo de desenvolvedores vem mantendo e aperfeiçoando o código fonte sob o nome de PostgreSQL. A versão 7.2.X do PostgreSQL apresenta as seguintes características:

- Portabilidade: atualmente existem versões para quase todos os sistemas operacionais;
- Suporte a SQL (compatível com SQL-92);
- Suporte a Herança entre Tabelas;
- Tipo de Dados Matriz (Tamanho Variável/Fixo);
- Tipos de Dados Geométricos - Ponto, Linha, Caminho, Polígono, Círculo;
- Suporte a Blobs;

- Possibilidade de criação de tipos de dados, funções e operadores definidos pelo usuário (mecanismo de extensibilidade);
- Linguagens procedimentais (PLSQL, PLTCL, PLPERL);
- Triggers e Regras (Rules);
- Controle de Concorrência e Transação;
- Métodos de indexação: árvores B, árvores R, HASH e GiST;
- Conectvidade através de interfaces C, C++, Java e ODBC entre outras
- Tabelas tamanho ilimitado, dependendo apenas do espaço em disco
- Campos de tamanho ilimitado (TOAST).

Entre as características do PostgreSQL mais importantes para este trabalho estão: mecanismo de extensão e a capacidade de trabalhar com dados de tamanho ilimitados. O fato de o PostgreSQL permitir trabalhar com dados de tamanho ilimitado é bastante interessante para o trabalho com dados geográficos que, normalmente, são bem grandes. A extensibilidade é de vital importância para o projeto e, portanto, merece uma atenção especial.

4.2 PostGIS

O PostGIS é uma extensão espacial gratuita e de código fonte aberto, que está sendo desenvolvida pela Refractions Research Inc (http://postgis.refractions.net), como um projeto de pesquisa em tecnologia de bancos de dados espaciais. Esta extensão está sendo construída sobre o PostgreSQL. Ela segue o padrão da SFSSQL, e possui o seguinte status, atualmente:

- Possui os tipos de objetos geométricos que podem ser representados e armazenados como um subconjunto do nível 3 da tabela apresentada na Seção 3.1.4:

  Point: (0 0 0)

  LineString: (0 0, 1 1, 1 2)

  Polygon: ((0 0 0, 4 0 0, 4 4 0, 0 4 0, 0 0 0), (1 1 0, ...), ...)

MultiPoint: (0 0 0, 1 2 1)

MultiLineString: ((0 0 0, 1 1 0, 1 2 1), (2 3 1, 3 2 1, 5 4 1))

MultiPolygon: (((0 0 0, 4 0 0, 4 4 0, 0 4 0, 0 0 0), (...), ...), ...)

GeometryCollection: (POINT(2 3 9), LINESTRING((2 3 4, 3 4 5))

- Possui um esquema para definição de metadados conforme a SFSSQL (representada pelas Tabelas 4.1 e 4.2:

spatial_ref_sys		
Attribute	**Type**	**Modifier**
srid	integer	NOT NULL PRIMARY KEY
auth_name	VARCHAR(256)	
auth_srid	integer	
srtext	VARCHAR(2048)	

TABELA 4.1 – Tabela de Metadados do Sistema de Coordenadas

geometry_columns		
Attribute	**Type**	**Modifier**
f_table_catalog	integer	NOT NULL PRIMARY KEY
f_table_schema	VARCHAR(256)	
f_table_name	integer	
f_geometry_column	VARCHAR(2048)	
coord_dimension	integer	
srid	integer	
type	VARCHAR(30)	

TABELA 4.2 – Tabela de Metadados das Colunas com Geometria

Para estas tabelas, encontra-se definida uma função, AddGeometryColumn('dbname', 'tablename', 'columngeomname', srid, 'postgis', ndimension), que é utilizada para criar os tipos de dados espaciais nas tabelas e automaticamente preencher as tabelas de metadado.

- Permite criação de índices espaciais através de uma R-Tree implementada sobre o GiST (discutido mais adiante);

- Apresenta alguns operadores métricos, como área e comprimento;

- Não apresenta operadores espaciais como os topológicos e os que geram novas geometrias como Buffer, ConvexHull e Interseção.

4.3 Metodologia

4.3.1 Definição dos Operadores Espaciais

Esta fase do projeto consiste no estudo e determinação dos operadores a serem implementados. Entre os possíveis operadores a serem implementados estão: Open(), Disjoint(), Touch(), Within(), Overlap(), Cross(), Intersects(), Contains(), Relate(), Distance(), Intersection(), Difference(), Union(), SymmetricDifference(), Buffer(), ConvexHull(). Como este trabalho é colaborativo com o PostGIS, a determinação dos operadores será feita em parceria com estes.

4.3.2 Implementação e Integração dos Operadores

Os operadores espaciais serão implementados de acordo com a matriz de 9-interseções dimensionalmente extendida. A implementação conta com o suporte dos algoritmos utilizados pelo SPRING e por uma API Java conhecida com Java Topology Suite (JTS) desenvolvida pela Vivid Solutions (http://www.vivid.com), conforme as especificações da SFSSQL. A implementação contará com o potencial de extensão do PostgreSQL, que está ligado ao fato de ele armazenar no catálogo do sistema não apenas informações sobre tabelas e colunas mas também sobre tipos de dados, funções e métodos de acesso definidos pelo usuário.

4.3.2.1 Extensão dos Tipos de Dados

Os tipos de dados a serem utilizados serão os definidos no PostGIS. A extensão de tipos de dados no PostgreSQL pode ser feita em linguagem C [20] através da definição de uma estrutura de dados. Esta estrutura é responsável pela representação do tipo em memória. Por exemplo, o trecho abaixo ilustra a definição de um tipo em C, chamado TeLinearRing :

```
/*---------------------------------------------------------------------
 * TeLinearRing -> Provides support for a 2D linear ring.
 *                 A linear ring is a 2D line (without self-intersections)
 *                 whose first point is the same as the last point.
 *---------------------------------------------------------------------*/
typedef struct
```

```
{
 int32 size;       //struct length, required for
//PostrgreSQL as varlena types
 int32 ncoords_;        //number of coords in the line
 TeCoord2D coords_[1];  //variable length array of Coords
} TeLinearRing;
```

Os tipos definidos pelo usuário podem ser de tamanho variável, como o do exemplo acima. Além da definição da estrutura, é necessário definir outras duas rotinas, que fazem a conversão do tipo para ASCII e vice-versa. Estas rotinas são conhecidas como funções de entrada e saída. Portanto, é necessário especificar uma representação textual externa para o dado, como por exemplo, para o tipo acima: [(X1, Y1), (X2, Y2), ..., (Xn, Yn)].

A função de entrada recebe uma string (na representação externa) como parâmetro e retorna a representação interna do tipo de dado. O trecho de código abaixo ilustra a definição de uma rotina de entrada para o tipo TeLinearRing.

```
PG_FUNCTION_INFO_V1(TeLinearRing_in);
Datum TeLinearRing_in(PG_FUNCTION_ARGS)
{
 char *str = PG_GETARG_CSTRING(0);
 TeLinearRing *ring;
 int ncoords;
 int size;
 char *s;
 if((ncoords = elements_in_TeLine2D_or_SimpleSet(str, RDELIM_TeLine2D)) <= 0)
   elog(ERROR, "Bad TeLinearRing external representation '%s'", str);
 size = offsetof(TeLinearRing, coords_[0]) + sizeof(ring->coords_[0]) * ncoords;
 ring = (TeLinearRing *) palloc(size);
 MemSet((char *) ring, 0, size);
 ring->size = size;
 ring->ncoords_ = ncoords;
 if((!TeLine2D_SimpleSet_decode(ncoords, str, &s, &(ring->coords_[0]),
          LDELIM_TeLinearRing, RDELIM_TeLinearRing)) || (*s != '\0'))
elog(ERROR, "Bad TeLinearRing external representation '%s'", str);
 if(!is_TeLinearRing(ring))
```

```
   {
    pfree(ring);
    ring = NULL;
    elog(ERROR, "In a TeLinearRing the first point must be the same as
                 the last point '%s'", str);
   }
 PG_RETURN_TeLine2D_P(ring);
}
```

A função de saída recebe o dado na representação interna e deve convertê-lo para a representação externa. Abaixo é ilustrada a função de saída para o tipo TeLinearRing.

```
PG_FUNCTION_INFO_V1(TeLinearRing_out);
Datum TeLinearRing_out(PG_FUNCTION_ARGS)
{
 TeLinearRing *ring = (TeLinearRing *)PG_GETARG_TeLinearRing_P(0);
 char *str;
 str = palloc(ring->ncoords_ * (P_MAXLEN + 3) + 2);
 if(!TeLine2D_SimpleSet_encode(ring->ncoords_, ring->coords_, str,
                               LDELIM_TeLinearRing, RDELIM_TeLinearRing,
                              "Unable to format TeLinearRing"))
   elog(ERROR, "Unable to format TeLinearRing");
 PG_RETURN_CSTRING(str);
}
```

Depois de criado o tipo, uma biblioteca compartilhada deve ser gerada para ser integrada dinamicamente ao servidor de banco de dados. Isto evita que o servidor tenha que ser interrompido. Depois, é necessário registrar o tipo através do comando SQL **CREATE TYPE**, informando as rotinas de conversão. Por exemplo:

```
/*********************************************
  TeLinearRing SQL statements
 *********************************************/
CREATE FUNCTION telinearring_in(opaque)
    RETURNS telinearring
    AS '/opt/tepgdatatypes.so'
    LANGUAGE 'c';
```

```
CREATE FUNCTION telinearring_out(opaque)
    RETURNS opaque
    AS '/opt/tepgdatatypes.so'
    LANGUAGE 'c';

CREATE TYPE telinearring
(
 alignment = double,
 internallength = VARIABLE,
 input = telinearring_in,
 output = telinearring_out,
 storage = main
);
```

4.3.2.2 Extensão das Funções e Operadores

Além da flexibilidade do sistema de tipos, o PotgreSQL permite a criação de métodos que operem sobre as instâncias dos tipos definidos. O trecho de código abaixo ilustra uma função cuja tarefa é calcular a área do tipo de dado TeLinearRing definido anteriormente.

```
/*---------------------------------------------------------------------
 * Area of a TeLinearRing
 *---------------------------------------------------------------------*/
PG_FUNCTION_INFO_V1(TeLinearRingArea);
Datum TeLinearRingArea(PG_FUNCTION_ARGS)
{
 TeLinearRing *r = (TeLinearRing *)PG_DETOAST_DATUM(PG_GETARG_DATUM(0));
 double area = 0.0;
 int    npoints = 0;
 int    loop;
 npoints = r->ncoords_;

 for(loop = 0; loop < (npoints - 1); loop++)
    {
```

```
     area += ((r->coords_[loop].x_ * r->coords_[loop + 1].y_) -
             (r->coords_[loop + 1].x_ * r->coords_[loop].y_)) ;
    }
 area *= 0.5;
 PG_RETURN_FLOAT4(area);
}
```

A função também deve ser colocada em uma biblioteca compartilhada para poder ser integrada ao servidor de banco de dados. Depois, é necessário registrar a função através do comando SQL **CREATE FUNCTION**, como mostrado abaixo:

```
CREATE FUNCTION area(telinearring)
RETURNS float4
AS '/opt/tepgfunctions.so', 'telinearringarea'
LANGUAGE 'c' with (isstrict);
```

Uma funcionalidade importante oferecida pelo PostgreSQL é que os nomes das funções podem ser sobrecarregadas desde que os parâmetros sejam de tipos diferentes. Depois de criada uma função é possível definir um operador para ela através do comando SQL **CREATE OPERATOR**:

```
CREATE OPERATOR op_name
(
 leftarg = tipo,
 rightarg = tipo,
 procedure = função,
 commutator = op_name
);
```

A importância da definição do operador está ligada ao otimizador de consultas que pode usar as informações contidas na definição do operador para decidir a melhor estratégia para a consulta (ou simplificação desta). Outra funcionalidade oferecida pelo PostgreSQL é a definição de funções agregadas, que podem ser construídas de forma análoga às funções sobre tipos, porém são registradas com o comando SQL **CREATE AGGREGATE**.

4.3.2.3 Extensão do Mecanismo de Indexação

Conforme dito anteriormente, o PostgreSQL possui três tipos de indexação (B-Tree, R-Tree, GiST) e um método de acesso HASH. Esses índices podem ser usados para os tipos de dados e funções definidos pelo usuário, bastando para isso registrar no catálogo do sistema as informações de operações necessárias para cada índice. Por exemplo, para um tipo que deseje ser indexado segundo a B-Tree, é necessário informar ao sistema as operações <, <=, =, >= e > para que o índice saiba como realizar buscas. No caso de tipos de dados espaciais, o mecanismo interessante de ser utilizado é a R-Tree, implementada segundo o algoritmo de Guttman. No entanto, a implementação do PostgreSQL apresenta uma restrição muito séria para os dados espaciais, ele só é capaz de indexar objetos com até 8 kbytes.

Uma segunda forma de definição de um índice espacial é através do **GiST**. Este mecanismo de indexação foi introduzido por Hellerstein, J. et al. [18] e implementado no PostreSQL. Atualmente, este índice é mantido no PostgreSQL por Teodore Signaev e Oleg Bartunov (http://www.sai.msu.su/ megera/postgres/gist) e não possui restrições de tamanho do dado a ser indexado. GiST é uma abreviação de Generalized Search Trees, consistindo em um índice (árvore balanceada) que fornece as funcionalidades tanto de uma B-Tree quanto de uma R-Tree e suas variantes.

A principal vantagem desse tipo de índice é a extensibilidade em relação aos dados e consultas sobre estes. Nesta árvore, as consultas são generalizadas de forma que as chaves de busca possam ser qualquer predicado arbitrário que valha para cada dado abaixo da chave. Os nós dessa árvore são muito parecidos com os das árvores B e R, no entanto, os elementos de cada nó são compostos pelos pares ¡predicado, ponteiro¿, onde o predicado é usado como chave de pesquisa e o ponteiro aponta para outros nós (no caso de nós internos) ou para os dados no banco de dados (caso dos nós folhas). Para implementar um determinado tipo de indexação (como uma B-Tree ou R-Tree) sobre o GiST, é necessário a implementação de seis métodos: Consistent(E, q), Union(P), Compress(E), Decompress(E), Penalty(E_1, E_2) e PickSplit(P).

4.3.3 *Driver* de Bancos de Dados TerraLib

Um dos objetivos da TerraLib é o desenvolvimento de aplicativos GIS baseados nos avanços tecnológicos dos sistemas de bancos de dados, especialmente os espaciais, realizando a completa integração dos tipos de dados espaciais dentro dos SGBDs.

Para realizar tal tarefa, ela fornece uma arquitetura que dá acesso direto aos dados que podem ser armazenados em diversos SGBDs, como Oracle Spatial, PostgreSQL e MySQL, possibilitando a criação e a manipulação de bancos de dados geográficos. Essa arquitetura fornece uma interface comum que possibilita ao desenvolvedor em não ter que se preocupar com os detalhes de cada SGBD. Isso permite que este enfoque nas funcionalidades que um GIS deve ter, como ferramentas para análise espacial, visualização gráfica dos dados geográficos e entrada de dados ao invés de se preocuparem com o gerenciamento dos dados. A arquitetura de construção de bancos de dados geográficos, apresentada na Figura 4.1, é composta pelos seguintes componentes:

- **Modelo de Dados:** composto por um conjunto específico de tabelas para a representação dos dados geográficos nos SGBDs;
- **Kernel:** composto pelas classes básicas (estruturas de dados) para representação em memória dos dados geográficos tanto no formato vetorial quanto matricial, por operadores topológicos e direcionais, classes de sistema de projeção e algoritmos;
- ***Drivers*:** formado por classes que fornecem uma interface comum e que dão o suporte básico para trabalhar com os dados geográficos nos SGBDs;
- **SGBDs:** a TerraLib permite a integração tanto com SGBDs Relacionais quanto com Objeto-Relacionais. Atualmente, ela possui interface com o SGBD relacional MySQL e com os objeto-relacionais Oracle Spatial e PostgreSQL. Os *drivers* são os responsáveis em manter essas interfaces.

Este trabalho propõe o desenvolvimento de um *driver* de banco de dados para o PostGIS em linguagem C++ [33]. A seguinte metodologia será empregada para isso:

- O suporte necessário para criação do modelo de dados da TerraLib será feito através do mapeamento entre os tipos de dados presentes no Kernel da biblioteca e os tipos de dados do PostgreSQL e do PostGIS;
- Os métodos para consultas espaciais empregarão o dialeto SQL fornecido pela extensão PostGIS e serão delegados para esta sua execução;
- Os índices espaciais serão construídos através do suporte GiST fornecido pelo PostGIS.

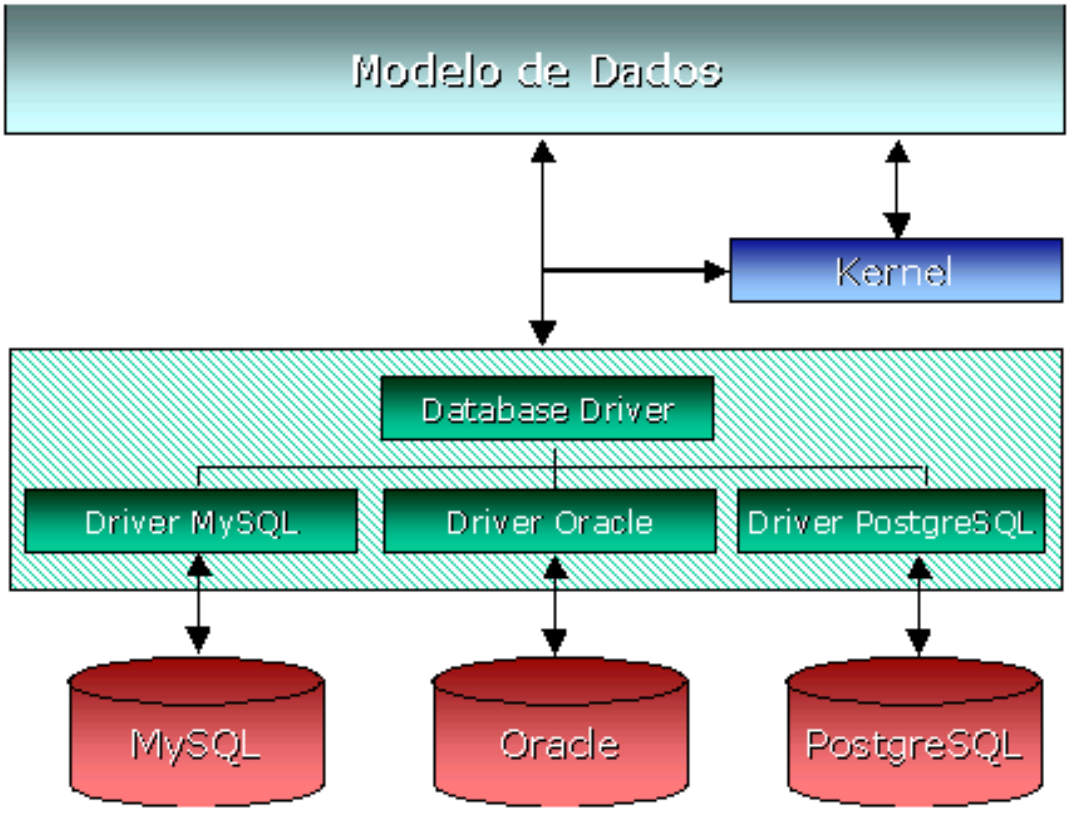

Fig. 4.1 – Arquitetura da TerraLib para construção de bancos geográficos

4.3.3.1 Aplicativo Spatial Database Explorer

Este aplicativo será construído com o suporte oferecido pela TerraLib, especialmente pelo *driver* com o PostGIS. Este software deverá apresentar as seguintes características:

- Interface com diferentes SGBDs: possibilitando que o usuário selecione o SGBD a ser utilizado, instanciando o *driver* correspondente e fazendo a conexão com o servidor de dados;
- Possibilidade de criação de tabelas com tipos de dados espaciais;
- Possibilidade de definição de índices espaciais;
- Visualização de dados geográficos: sua componente espacial será apresentada em um canvas e seus atributos alfanuméricos através de uma tabela;

- Possibilidade de execução de consultas espaciais;
- Possibilidade de importação de dados de arquivos de formato proprietários para o SGBD, a fim de possibilitar o povoamento da base de dados para testes, pois a interface gráfica não apresentará recursos para entrada de dados;
- Recursos para apresentar ao usuário o tempo levado para executar determinada tarefa. Por exemplo: tempo levado para indexar a base de dados, para execução de uma determinada consulta ou tempo levado para a realização de importação de dados.

A seguinte metologia será empregada na sua elaboração:

- Uso da linguagem de programação C++;
- Aplicação de Programação Genérica [1] e Design Patterns [13] no desenvolvimento;
- Projeto em UML [24];
- Interface gráfica com usuário desenvolvida em Qt [15].

4.4 Resultados Esperados

Como resultados esperados para o desenvolvimento deste trabalho temos:

- Desenvolvimento de operadores espaciais eficientes que integrados ao PostGIS o tornem uma solução "open source" alternativa a produtos comerciais;
- Mostrar o potencial da nova geração de SGBD-OR para gerenciamento de dados geográficos.

4.5 Cronograma

	jun	jul	ago	set	out	nov	dez	jan/2003
definição dos operadores	X							
implementação dos operadores	X	X	X	X	X	X	X	
implementação do *driver*					X	X	X	
implementação do SDE					X	X	X	X
dissertação					X	X	X	X

TABELA 4.3 – Cronograma - Junho/2002 a Janeiro/2003

REFERÊNCIAS BIBLIOGRÁFICAS

[1] M. H. Austern. *Generic Programming and the STL*. Addison-Wesley, Massachusetts, 1999.

[2] R. R. Ciferi and A. Salgado. Análise de eficiência de métodos de acesso espaciais em termos da distribuição espacial dos dados. *III Workshop Brasileiro de Geoinformática – GeoInfo*, pages 79–86, 2001.

[3] G. Câmara, M. A. Casanova, A. S. Hemerly, G. C. Magalhães, and C. M. B. Medeiros. *Anatomia de Sistemas de Informação Geográfica*. Unicamp, Campinas, 1996.

[4] G. Câmara, R. C. M. Souza, B. M. Pedrosa, L. Vinhas, A. M. V. Monteiro, J. A. Paiva, M. T. Carvalho, and M. Gatass. Terralib: Technology in support of gis innovation. *II Workshop Brasileiro de Geoinformática – GeoInfo*, 2000.

[5] G. Câmara, L. Vinhas, R. C. M. Souza, J. A. Paiva, A. M. V. Monteiro, M. T. Cravalho, and B. Raoult. Design patterns in gis development: The terralib experience. *III Workshop Brasileiro de Geonformática – GeoInfo*, pages 95–101, 2001.

[6] C. J. Date. *Introdução a Sistemas de Bancos de Dados*. Campus, Rio de Janeiro, 1990.

[7] C. Davis and G. Câmara. *Arquiteturas de Sistemas de Informação Geográfica*, chapter 3. INPE, São José dos Campos, 2 edition, 2001.

[8] J. R. Davis. Ibm's db2 spatial extender: Managing geo-spatial information within the dbms. [online]. <*http://www.ibm.com.br*>, 1998.

[9] M. J. Egenhofer, E. Clementini, and P. Di Felice. Topological relations between regions with roles. *International Journal of Geographical Information Systems*, 8(2):129–142, 1994.

[10] M. J. Egenhofer and R. Franzosa. On the equivalence of topological relations. *International Journal of Geographical Information Systems*, 9(2):133–152, 1995.

[11] M. J. Egenhofer and J. R. Herring. Categorizing binary topological relations between regions, lines, and points in geographic databases. Technical report, University of Maine, Orono, 1991.

[12] A. U. Frank. Requirements for database systems suitable to manage large spatial databases. *International Symposium on Spatial Data Handling*, pages 38–60, 1984.

[13] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Padrões de Projeto: Soluções reutilizáveis de software orientado a objetos*. Bookman, PortoAlegre, 2000.

[14] H. Garcia-Molina, J. D. Ullman, and J. Widom. *Implementação de Sistemas de Bancos de Dados*. Campus, Rio de Janeiro, 2001.

[15] A. Griffith. *KDE 2/Qt Programming Bible*. IDG Books, Foster City, 2001.

[16] R. H. Güting. An introduction to spacial database systems. *VLDB Journal*, 3(4), Oct 1994.

[17] A. Guttman. R-trees: A dynamic index structure for spatial search. *ACM SIGMOD*, pages 47–57, Jun 1984.

[18] J. M. Hellerstein, J. F. Naughton, and A. Pfeffer. Generalized serch trees for databases systems. *VLDB Journal*, Jun 1995.

[19] IBM, <http://www.ibm.com.br>. *IBM DB2 Spatial Extender: User's Guide and Reference. [online]*, Jun 2001.

[20] B. W. Kernighan and D. M. Ritchie. *C – A Linguagem de Programação Padrão ANSI*. Campus, New York, 1990.

[21] C. M. B. Medeiros and F. Pires. Databses for gis. *ACM SIGMOD*, pages 107–115, 1994.

[22] C. Murray. *Oracle Spatial User's Guide and Reference, Release 9.0.1. [online]*. Oracle Corporation, <http://www.oracle.com>, 2001.

[23] Inc. OpenGIS Consortium. Opengis simples features specification for sql revision 1.1. [online]. *<http://www.ogis.org>*, May 1995.

[24] M. Page-Jones. *Fundamentals of Object-Oriented Design in UML*. Addison-Wesley, 2000.

[25] P. Ramsey. Postgis manual. [online]. *<http://postgis.refractions.net/documentation>*, Jan 2002.

[26] S. Ravada and J. Sharma. Oracle 8i spatial: Experiences with extensible databases. *SSD99*, 1999.

[27] P. Rigaux, M. Scholl, and A. Voisard. *Spatial Databases: with application to GIS*. Morgan Kaufmann Publishers, San Francisco, 2001.

[28] H. Samet and W. G. Aref. *Spatial Data Models and Query Processin*, chapter 17. Addison Wesley/ACM Press, Massachusetts, 1994.

[29] S. Shekar, S. Chawla, S. Ravada, A. Fetterer, X. Liu, and C. Lu. Spatial databases – accomplishments and research needs. *IEEE Transactions on Knowledge and Data Engineering*, 11(1), Jan 1999.

[30] A. Silberschatz, H. F. Korth, and S. Sudarshan. *Sistemas de Bancos de Dados*. MAKRON Books, São Paulo, 1999.

[31] M. Stonebraker. *Object-Relational DBMSs: The Next Great Wave*. Morgan Kaufmann Publishers, San Francisco, 1996.

[32] M. Stonebraker, L. A. Rowe, and M. Hirohama. The implementation of postgres. *IEEE Transactions on Knowledge and Data Engineering*, 2(1):125–142, Mar 1990.

[33] B. Stroustrup. *A Linguagem de Programação C++*. Bookman, Porto Alegre, 2000.