UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA

CURSO CIÊNCIA DA COMPUTAÇÃO
TURMA 2005.2

Estendendo a Ferramenta JUMP Para Realização de Consultas SQL *Ad Hoc*

Autor

Gilberto Antonio da Silva Júnior (gasj2@cin.ufpe.br)

Orientadora

Valéria Cesário Times (vct@cin.ufpe.br)

Recife, Agosto de 2005.

A minha vovó neta

Agradecimentos

Gostaria de agradecer primeiramente aos meus pais, grandes lutadores que sempre batalharam para dar o melhor possível a mim e a meus irmãos. Sem eles, acho que não estaria escrevendo este trabalho nem tão cedo. Pai, Mãe: Muito Obrigado!

Agradeço ao amor da minha vida, Renatta que me agüentou esse (tempo todo não sei como), sempre me apoiou nas minhas decisões e sempre esteve presente de uma forma ou de outra em toda essa longa jornada que está quase acabando. Amor: Te amo!

Agradeço também aos amigos que conheci e que me ajudaram bastante em todo esse caminho. Foram muitos, espero não esquecer ninguém: Eduardo (Lemco), Diego, Pedroka, Ênio, Aércio, Afonso, Fausto, Giovani, Nem (do cachorro quente), Alexandre (ATV), Daniel Agra e Eudes. Galera: Valeu!

Aos professores, em especial a Clilton Galamba, Fernando Fonseca, Valéria e ao grande Ruy Guerra que me ensinou que com fé e muita sorte agente passa por qualquer coisa.

A Alexandre Lemos da Eólica onde fiz a minha iniciação científica durante mais de dois anos por sua paciência, pelo conhecimento passado, sua compreensão pela minha falta de tempo e principalmente pela confiança e empolgação com o meu trabalho. Obrigado Alexandre.

A Ângelo Roque, o banqueiro, por ter sido um grande amigo, um cara com um grande conhecimento de tudo. Sempre compreendeu quando eu não podia ajudar por estar ocupado. Me ajudou em vários momentos difíceis. Ângelo, agradeço a Deus por amigos como você.

Finalmente, agradeço a Deus pela minha vida de muito trabalho e estudo, mas feliz. Por ter pessoas tão maravilhosas perto de mim e pelas dificuldades que me fazem aprender a andar pelo caminho certo, mesmo que nem sempre o mais fácil.

Resumo

Existia no Projeto **GOLAPA** [FTS01, GHP03] do Centro de Informática da UFPE a necessidade de uso de uma ferramenta que pudesse exibir dados geográficos e analíticos na Web. Para estes fins, era necessária a realização de um estudo para identificar qual a melhor ferramenta existente atualmente.

Através de uma pesquisa entre os sistemas disponíveis que poderiam suprir esta necessidade, a ferramenta **JUMP** (Java Unified Mapping Platform) foi a que mais se mostrou capaz de realizar tais necessidades. O JUMP é um poderoso *framework* para visualização e manipulação de dados de natureza espacial distribuído sob a licença **GPL** [GNU03] e construído de forma a incentivar sua expansão (através de *PlugIns*) pela comunidade de desenvolvedores e usuários do mesmo.

Um problema atual do JUMP é que não é possível ao usuário realizar consultas *ad hoc*, sendo possível apenas selecionar uma das tabelas existentes que se deseja visualizar. A proposta deste trabalho, além da pesquisa realizada, é construir uma extensão para o mesmo de forma a possibilitar que o usuário possa fazer consultas de forma livre a qualquer banco de dados do SGBD PostgreSQL [PG01].

Índice

Índice de Figuras

Lista de Quadros

Principais Abreviaturas

API	*Application Program Interface*
GML	*Geography markup Language*
GOLAPA	*Geographical Online Analytical processing Architecture*
JTS	*Java Topology Suite*
OGIS	*OpenGIS*
OGC	*OpenGIS Consortium*
SFSSQL	*Simple Feature Specification for SQL*
SGBD	*Sistemas de Gerenciamento de Bancos de Dados*
SI	*Sistemas de Informação*
SIG	*Sistemas de Informação Geográfica*
SQL	*Structured Query Language*
SSL	*Secure Socket Layer*
TCP	*Transmission Control Protocol*
UML	*Unified Modeling Language*
WMS	*Web Map Service*

1 Introdução

Este capítulo indica as principais motivações para a realização deste trabalho, as pesquisas realizadas, os objetivos e, finalmente, mostra como está estruturado o restante do mesmo.

1.1 Motivação

Atualmente, existe um grande interesse tanto do meio acadêmico como de grandes corporações no estudo e utilização dos chamados SIG (Sistemas de Informação Geográfica) [CCH+96]. Um SIG é um sistema computacional que permite captar, modelar, manipular, recuperar, consultar, analisar e apresentar soluções com dados geograficamente referenciados. A consulta aos dados pode ser feita sobre os atributos alfanuméricos ou sobre sua localização. Uma consulta sobre atributos alfanuméricos responde questões do tipo “quais cidades possuem mais de 50.000 habitantes?”. Já uma consulta espacial responde questões como “quais são os postos de gasolina mais próximos da minha residência, num raio de 2 km?”.

Existem diversos tipos de sistemas que manipulam dados espaciais, como os sistemas de cartografia automatizada e os sistemas do tipo CAD (Computer Aided Design). Entretanto, os SIG se diferenciam desses sistemas por dois motivos principais. Primeiro, por sua capacidade de modelar os relacionamentos espaciais (topológicos) entre as representações espaciais do que existe no mundo real. Segundo, por permitir a realização de complexas operações de análise espacial.

Várias ferramentas estão disponíveis para realizar estas operações, algumas delas são do tipo *Open Source*, enquanto outras são soluções proprietárias. Foram estudadas algumas das ferramentas *Open Source* disponíveis atualmente e a mais adequada para satisfazer os requisitos do projeto foi o **JUMP** (Java Unified Mapping Platform) devido, principalmente, ao fato do mesmo poder utilizar o PostgreSQL e a possibilidade de ser estendido através de PlugIns.

Existe, porém, uma limitação do JUMP, pois o mesmo não permite a realização de consultas SQL de forma livre (*ad hoc*) sobre os dados contidos em um SGBD. Isto, de certa maneira, causa uma dificuldade a mais para as

pesquisas do projeto, pois muitas vezes as mesmas necessitam de resultados bastante específicos, o que só pode ser alcançado com consultas também específicas.

1.2 Objetivos

O objetivo principal do trabalho é estender as capacidades do JUMP através de um *PlugIn* para que seja possível a realização de consultas *ad hoc* pelos usuários do mesmo. Tal extensão será de grande utilidade para os usuários, pois possibilita uma grande flexibilidade no momento de realizar a obtenção dos dados contidos em bases de dados através de uma simples consulta SQL.

1.3 Estrutura

Para alcançar os objetivos almejados, bem como para apresentar alguns conceitos e tecnologias relacionadas a este trabalho, o restante do presente documento está organizado da seguinte forma:

Capítulo 2 – Tecnologias Envolvidas: Este capítulo descreve as tecnologias relacionadas com o presente trabalho ao mesmo tempo em que procura justificar a escolha do JUMP e do PostgreSQL como ferramentas ideais para a realização do mesmo.

Capítulo 3 – Trabalhos Correlatos: Aqui serão apresentadas as principais ferramentas SIG para desktop disponíveis atualmente. Com isso, será possível analisar as vantagens e desvantagens da cada uma e o motivo da escolha da ferramenta JUMP para o desenvolvimento do trabalho.

Capítulo 4 – Implementação: Este capítulo descreve o resultado prático do trabalho. A implementação de uma extensão que permite aos usuários do JUMP realizar consultas SQL *ad hoc* em bancos de dados PostgreSQL.

Capítulo 5 – Conclusões e Trabalhos Futuros: Este capítulo tem como objetivo apresentar algumas considerações finais sobre os principais tópicos abordados neste trabalho, incluindo as contribuições alcançadas e indicações para trabalhos futuros.

2 Tecnologias Envolvidas

Este capítulo descreve as tecnologias relacionadas com o presente trabalho, ao mesmo tempo em que procura justificar a escolha do JUMP e do PostgreSQL como ideais para a realização do mesmo.

2.1 SIG

Em diversas aplicações, Sistemas de Informação (SI) são usados progressivamente como uma maneira de gerenciar, recuperar e armazenar grandes quantidades de informação que são tediosas de manipular manualmente. Dados espaciais não são uma exceção. Tais dados podem ser manipulados por ferramentas conhecidas como Sistemas de Informação Geográfica (SIG) [Cla97, MGR91, Dem97], os quais são utilizados para aquisição, pré-processamento, gerenciamento, manipulação e análise, e apresentação de resultados de dados geograficamente referenciados. Os SIG são uma ferramenta importante para áreas que necessitam de aplicações computacionais capazes de manipular e analisar informações baseadas em sua localização no espaço. No contexto de Suporte à Decisão, os SIG são usados em diversas áreas de aplicação tais como planejamento urbano e rural, controle ambiental, gerenciamento de serviços de utilidade pública, e administração de recursos naturais [Ylu00].

Visto que um SIG trabalha com dados espaciais que são gráficos por natureza o mesmo necessita ter uma ótima interface gráfica com o usuário (GUI). Para a realização de análises em um SIG, normalmente são disponibilizados aos usuários, mapas que demonstram visualmente o comportamento dos dados armazenados em um banco de dados geográfico.

Em [JS01], o autor cita que em um SIG, a idéia de sobreposição de mapas é implementada empregando-se o conceito de sobreposição de temas ou camadas, no qual, para um mesmo espaço geográfico, podem ser criados diversos temas de dados geo-referenciados, um para cada acontecimento geográfico a ser representado. Esta abordagem facilita análises sobre os dados geo-referenciados, pois permite a combinação de dois ou mais temas para o processamento de uma determinada consulta [Fid03]. Na figura 2-1, pode ser visualizado um exemplo que representa a sobreposição de temas, conceito

amplamente utilizado nas consultas sobre dados espaciais e comumente definido como visão geográfica.

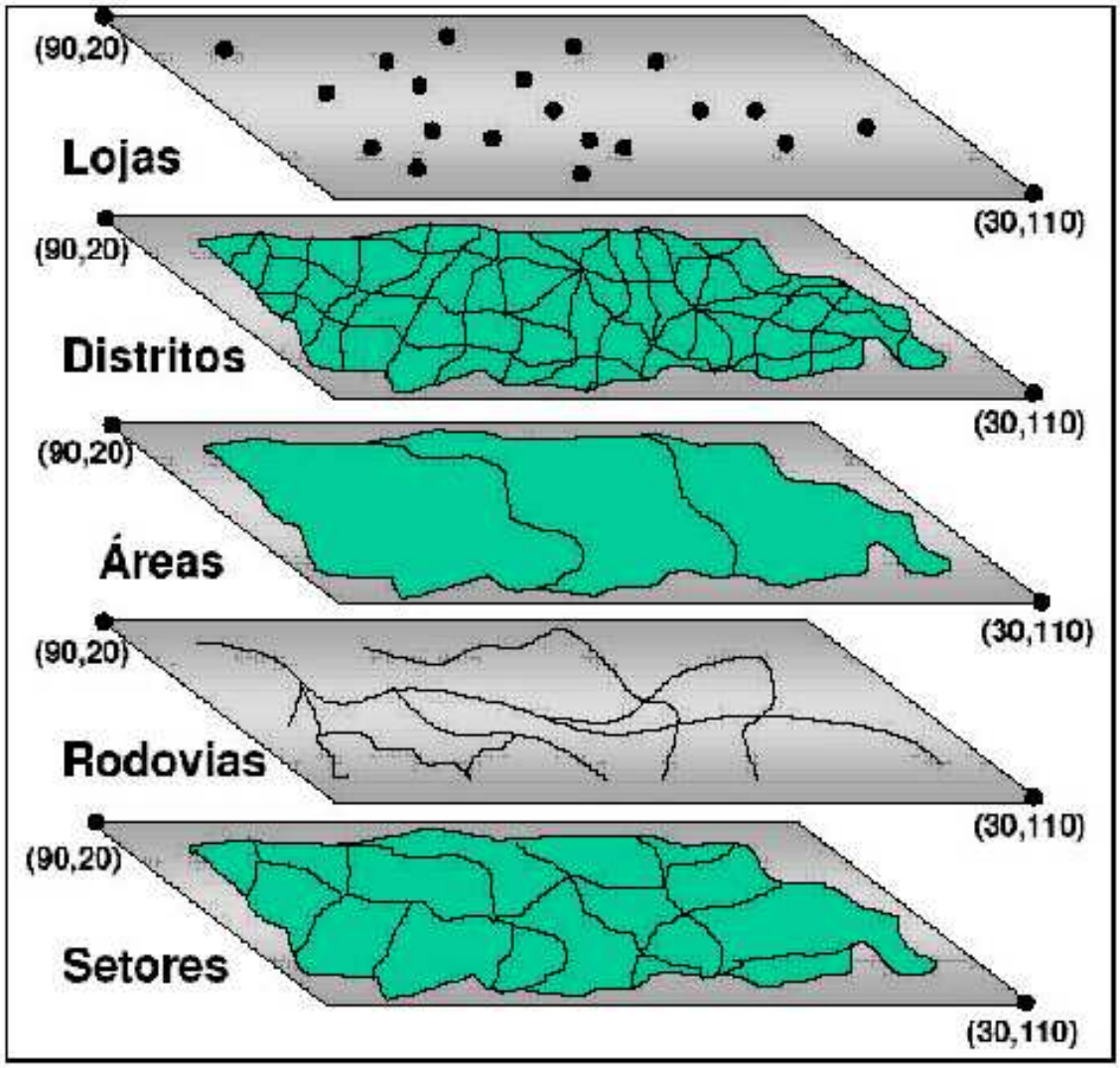

Figura 2-1 Exemplo de sobreposição de temas (extraído de [Fid03])

Em [CCH+96], o autor discorre sobre uma lista de componentes básicos de um SIG. A estrutura apresentada na Figura 2-2 representa o relacionamento existente entre estes componentes. Em um nível mais externo, se encontra a interface com o usuário, a qual disponibiliza a forma como o sistema é operado. No nível intermediário, se encontram os módulos responsáveis pelo processamento dos dados espaciais, incluindo entrada, edição, análise e visualização. Em um nível mais interno, se encontra o sistema de

gerenciamento de banco de dados geográficos, o qual deve suportar o armazenamento e recuperação dos dados espaciais. A seguir, cada um desses componentes é descrito brevemente.

- **Interface com Usuário**: Para substituir as antigas interfaces em modo texto, as quais eram baseadas em comandos digitados pelo usuário, pouco interativas e complexas para o mesmo, cada vez mais são disponibilizadas interfaces gráficas para manipulação das funcionalidades oferecidas pelos SIG, as quais são constituídas de menus, caixas de diálogo, e representação gráfica de operadores;

- **Entrada e Integração de Dados**: Compreende o conjunto de operações a serem aplicadas antes da utilização efetiva dos dados. Normalmente, este componente abrange funcionalidades que permitem a entrada de dados através de sua digitalização, leitura dos dados de forma digital, e procedimentos para validação dos dados e tratamento de erros oriundos da captura dos dados geo-referenciados;

- **Consulta e Análise Espacial**: Este é um dos mais importantes componentes de um SIG. Dentre as funcionalidades oferecidas por este componente, estão a seleção e pesquisa sobre informações geográficas, transformações de escala ou projeção, sobreposição de camadas de dados e execução de operações espaciais;

- **Visualização e Plotagem**: É responsável pela apresentação gráfica dos resultados de consultas e análises espaciais de maneira que o usuário possa interpretar facilmente estes dados.

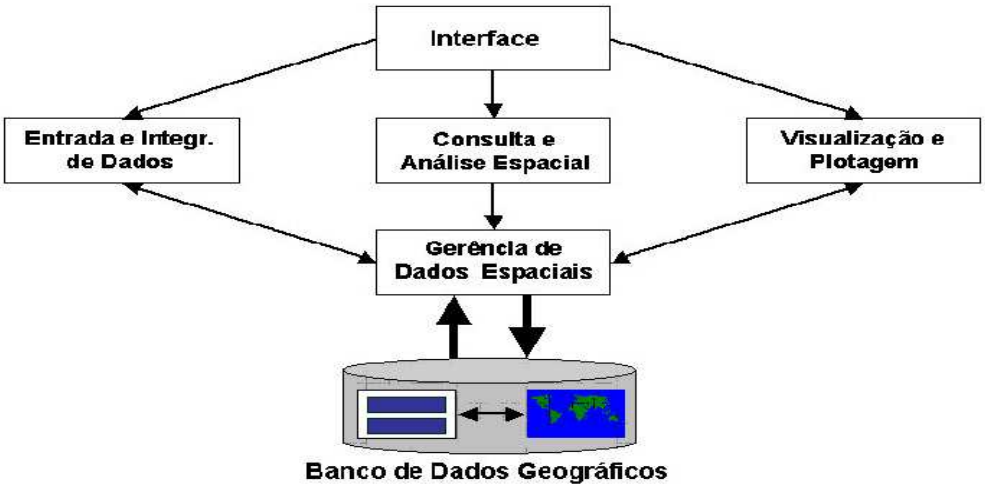

Figura 2-2 Componentes de um SIG (adaptado de [CCH+96])

2.2 OpenGIS

O OpenGIS (OGIS) [OGC01] é uma associação formada por organizações públicas e privadas envolvidas com SIG, dedicada à criação e gerenciamento de uma arquitetura padrão para geoprocessamento.

Uma das propostas do OpenGIS 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” [GR02].

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 estende os tipos da SQL. Como este trabalho consiste no uso de uma extensão com tipos de dados espaciais que estendem a SQL, apenas os detalhes do segundo modelo serão apresentados.

2.2.1 Modelo de Dados das Geometrias

A Figura 2-3 ilustra a hierarquia de tipos definida pela especificação do OGIS para dados geométricos. Esta hierarquia é definida em um modelo que 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.

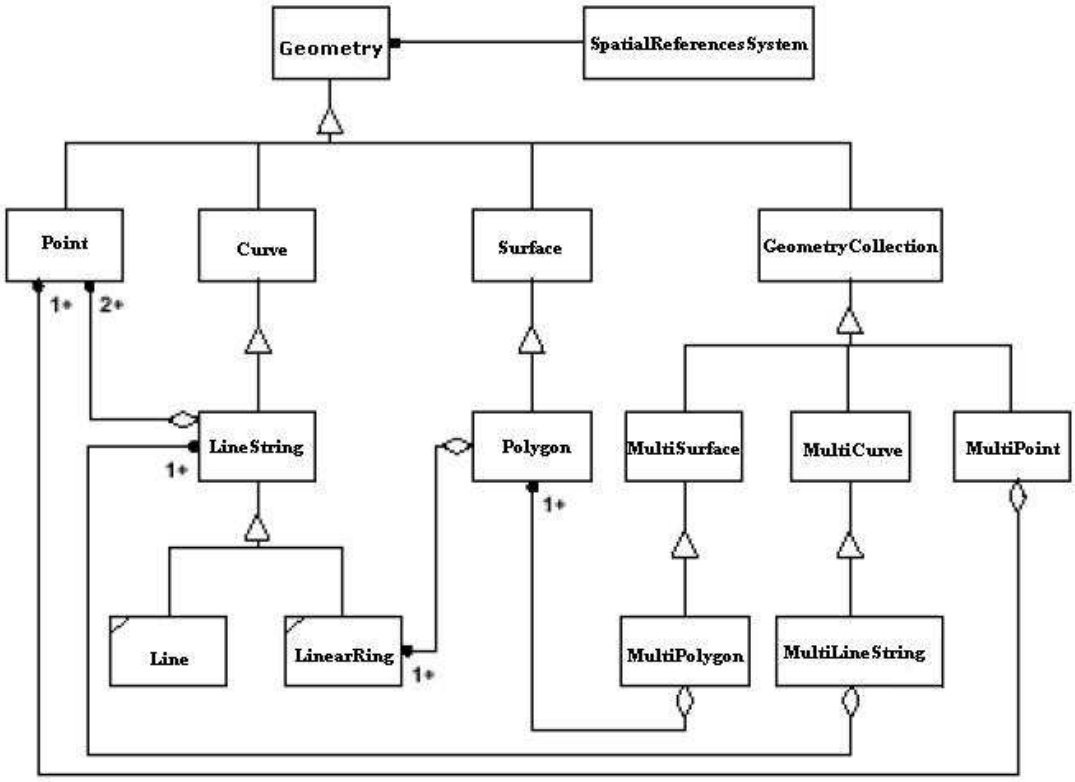

Figura 2-3 Hierarquia de classes das geometrias Fonte: [OGC01]

2.2.2 Operadores para Relacionamentos Topológicos

Os operadores topológicos 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 entre interseção de fronteira, interior e exterior) [GR02]. 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 Estendido [OGC01].

2.3 JUMP (Java Unified Mapping Platform)

Aqui procuramos descrever as principais características do sistema alvo deste trabalho, suas vantagens, limitações e sua arquiteura.

2.3.1 Visão geral

Este sistema é um *Framework* **Java** para o desenvolvimento de aplicações de SIG. Foi desenvolvido por uma empresa canadense chamada **Vivid Solutions** [VIV01]. Possui um ambiente gráfico bem amigável, excelente documentação e uma grande facilidade de programar novas funcionalidades. Nesse ambiente orientado a objeto, uma característica muito interessante é a flexibilidade de rodar em qualquer plataforma (característica da linguagem **Java**). Internamente, esse *Framework* é composto por uma biblioteca denominada JTS (Java Topology Suíte), uma poderosa biblioteca para análises espaciais sobre geometrias em 2D (também desenvolvida pela **Vivid Solutions)** que contempla inúmeros operadores topológicos e segue a especificação SFSSQL (Simple Feature Specification for SQL) [OGC01], sendo esta responsável pelas análises vetoriais presentes no JUMP.

Dentre as características técnicas, podemos citar:

- Possui várias funcionalidades para manipulação de feições (visualização, edição e criação);
- Interface com o usuário amigável através do *JUMP Workbench* (ver figura 2-4), uma interface gráfica extensível para visualização e manipulação de dados espaciais;
- Uma *API* para execução de tarefas básicas de manipulação de dados espaciais, incluindo entrada e saída de dados, indexação espacial e funções de análise, entre outras;
- Oferece suporte nativo a arquivos em formato ESRI® Shapefiles e GML;

- Permite conexão a servidores WMS;
- Suporte ao PostGIS através de *PlugIn*;

Vários *PlugIns* para o JUMP estão sendo disponibilizados livremente na Internet, permitindo expandir as funcionalidades da ferramenta. *PlugIns* são classes carregadas dinamicamente que têm acesso à maioria das funções internas do Workbench. Estes podem ser colocados no menu do JUMP e manipular e criar objetos do Workbench quando for necessário. Além disso, podem receber entradas de dados do usuário utilizando caixas de diálogo ou mesmo através da seleção de componentes visuais (por exemplo, pode-se selecionar uma camada na lista de camadas).

Uma limitação do JUMP é o fato de não possuir suporte para dados matriciais através da leitura de arquivos locais. Para se trabalhar com dados matriciais, é necessário fazer uma conexão com um servidor WMS (Web Map Service) no qual esteja o banco de imagens (arquivos matriciais geo-referenciados). Outro ponto negativo é o fato de não existir suporte interno para tratamento das projeções. Como a maioria dos usuários de SIG não trabalha com mudanças de projeção, dando ênfase às análises, isso não dificulta os trabalhos mais usuais.

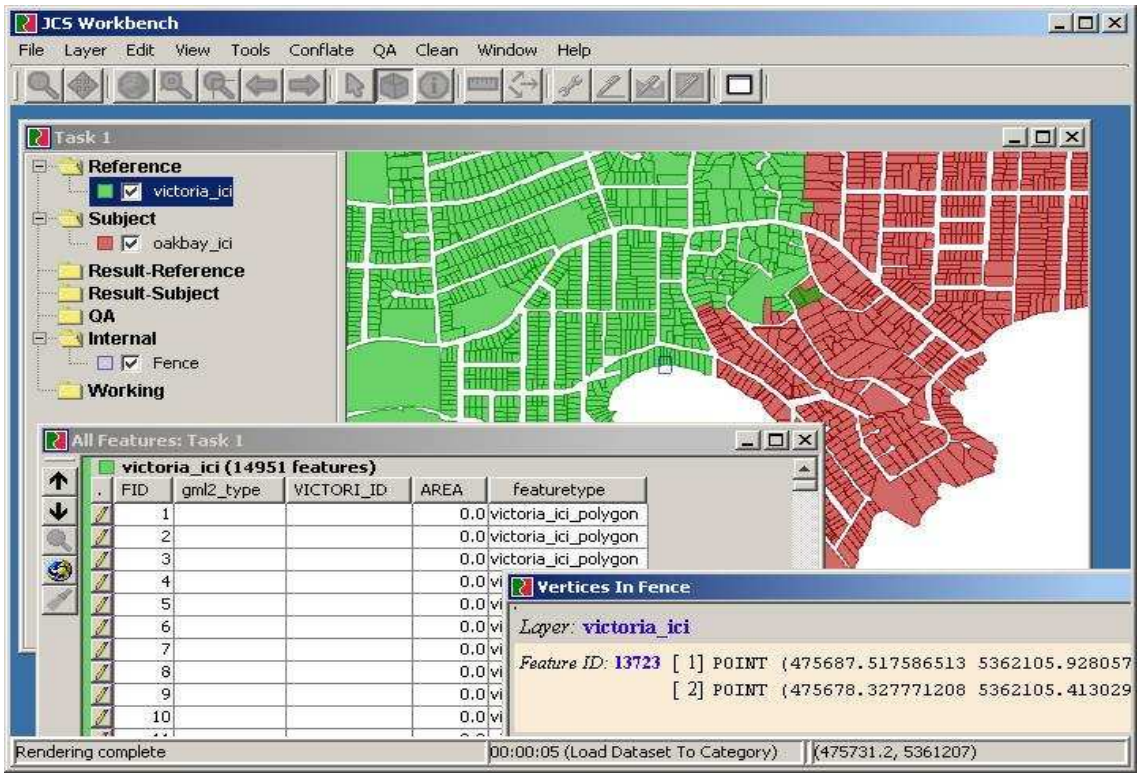

Figura 2-4 JUMP Workbench

2.3.2 Arquitetura

A arquitetura do JUMP é modular, reutilizável, e personalizável. As principais funções estão disponíveis por meio de uma *API* para facilitar o uso em outras aplicações. O *Workbench* fornece um *framework* e uma *API* permitindo que extensões possam ser adicionadas facilmente. A figura 2-5 ilustra a arquitetura do JUMP. O Quadro 2-1 traz informações sobre o site mantenedor do JUMP, a linguagem em que foi programado, o seu tipo de licença e os padrões OGC (OpenGIS Consortium) seguidos pelo mesmo.

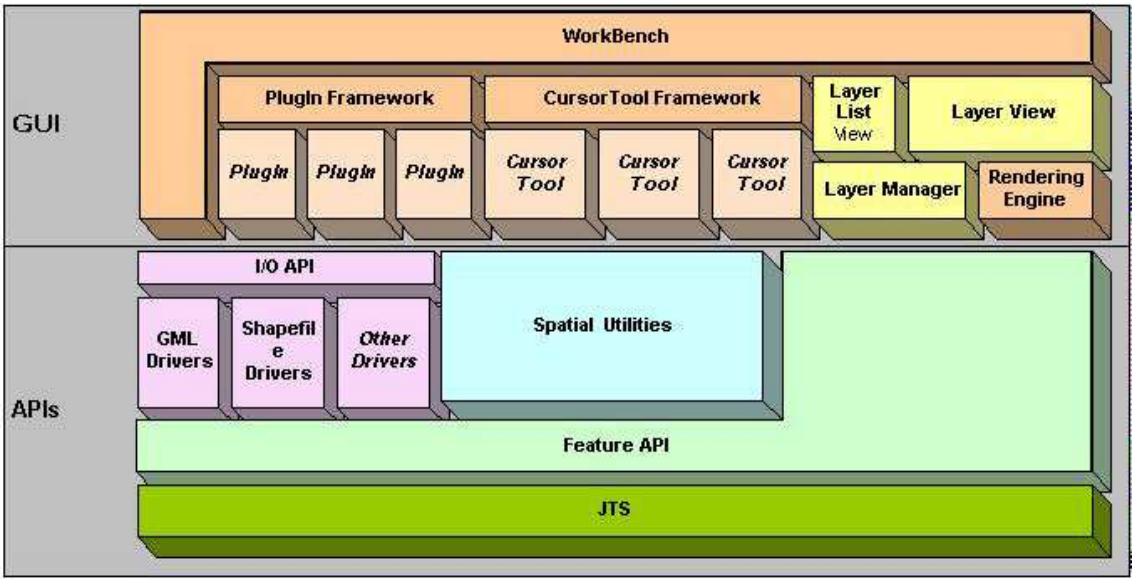

Figura 2-5 Arquitetura do JUMP

Quadro Resumido	
Site Principal:	http://www.jump-project.org/
Site Secundário:	http://www.vividsolutions.com/jump/
Mantenedor (responsável):	Vivid Solutions e Refractions Research
Linguagem (código-fonte):	Java
Licença:	GPL
Padrões OGC:	SFSSQL (implementação parcial através do *plug-in* de conexão ao PostGIS e implementação total através da JTS), WMS e GML

Quadro 2-1 Quadro resumido – JUMP (adaptado de [GEO01])

2.4 PostgreSQL

Apesar do JUMP poder se comunicar com qualquer banco que seja suportado pelo JDBC [JDBC01], o SGBD PostgreSQL foi escolhido por já existir um *PlugIn* para o JUMP que se comunicava com o mesmo. Além diso, trata-se de SGBD completo, *Open Source*, robusto e bastante utilizado dentro da comunidade acadêmica.

2.4.1 Visão geral

O **PostgreSQL** [PG01] é um banco de dados objeto-relacional. Seu grande triunfo é possuir recursos comuns a banco de dados de grande porte, o que o deixa apto a trabalhar, inclusive, com operações de missão crítica. Além disso, trata-se de um banco de dados versátil, seguro, gratuito e de código aberto.

É possível utilizar o PostgreSQL em vários sistemas operacionais, dentre os quais o Windows, Linux, BSD ou qualquer sistema compatível com as especificações POSIX (Portable Operating System Interface). Segundo informações do site oficial, o PostgreSQL permite a criação de uma base de dados de tamanho infinito. Cada tabela pode ter até 16 TB (1 terabyte = 1024 gigabytes), sendo que cada linha pode ter até 1,6 TB e cada campo 1 GB. O banco de dados ainda conta com conexões SSL (Secure Socket layer), *triggers*, integridade referencial, entre outros recursos, além de ser compatível com uma série de linguagens, tais como PHP, Python, Java e Pearl [PG01].

O uso do PostgreSQL é muito amplo, pois várias empresas perceberam que ele possibilita a criação de bases de dados complexas sem a necessidade de investir na aquisição de licenças. Outra vantagem, é que o PostgreSQL possui uma documentação muito abrangente, o que permite suporte adequado às necessidades dos usuários e administradores.

2.4.2 Arquitetura

Um único processo **postmaster** espera por conexões TCP (Transmission Control Protocol) ou Unix Sockets, e inicia um processo **postgres** para cada conexão. Assim sendo, o PostgreSQL utiliza naturalmente múltiplos processadores, mas para usuários simultâneos, não para acelerar um único comando SQL. Dois processos postgres estão sempre ativos, mas não atendem a conexões. Eles cuidam da gravação física de blocos de estruturas de log ou tabelas, e da manutenção de estatísticas. Todos os processos postgres compartilham duas áreas de memória: **O buffer cache** e o **write-ahead log**. O primeiro, armazena blocos lidos (ou modificados) de tabelas e índices, enquanto o segundo armazena temporariamente, o log de transações, até que ele possa ser armazenado em disco. Além disso, cada processo postgres tem uma área individual para operações de ordenação (**sort buffer**) [PG01]. A arquitetura do PostgreSQL pode ser observada na Figura 2-6.

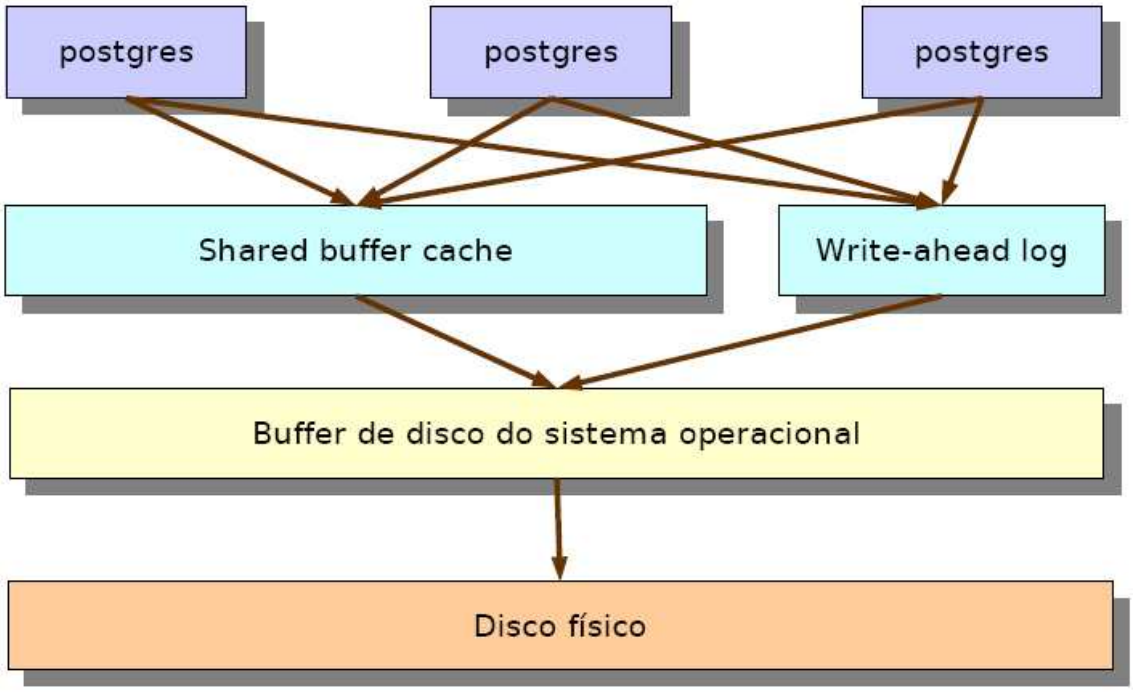

Figura 2-6 Arquitetura do PostgreSQL

2.4.3 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 os seguintes recursos atualmente [GR02]:

- Possui tipos de objetos geométricos que podem ser representados e armazenados:
  - Point;
  - LineString;
  - Polygon;
  - MultiPoint;
  - MultiLineString;
  - MultiPolygon;
  - GeometryCollection;
- Possui um esquema para definição de metadados conforme a SFSSQL;
- Permite criação de índices espaciais através de uma *R-Tree* implementada sobre o *GiST*;
- 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.

2.5 *Conclusão*

Após a pesquisa realizada entre os principais softwares desktop para análise e manipulação de dados espaciais de sistemas SIG (uma análise mais detalhada dos mesmos encontra-se no Capítulo 3: Trabalhos Correlatos), chegamos à conclusão que a solução que melhor atende às expectativas de um software simples, com uma interface amigável, *open source*, extensível e com acesso a um banco que dados com suporte a dados espaciais foi o **JUMP** da Vivid Solutions. Podemos listar alguns dos pontos de destaque do mesmo:

- Desenvolvido na linguagem JAVA, sendo esta bastante difundida e utilizada em várias áreas do conhecimento humano;
- Atende ao padrão SFSSQL do OpenGIS Consortium;
- Permite facilmente a extensão das suas funcionalidades através de plug-ins;
- Possui interface gráfica com o usuário;
- Documentação completa disponível;
- Oferece uma lista de discussão para desenvolvedores e usuários.

O quadro 2-2 exibe um comparativo entre o JUMP e os principais pontos relevantes dos SIG estudados neste trabalho.

(A - JUMP, B - Thuban, C - GRASS, D - Terraview, E - QGIS)

Quadro comparativo entre os sistemas estudados					
	A	**B**	**C**	**D**	**E**
Interface gráfica com o usuário	**S**	**S**	**N**	**S**	**S**
Obedece aos padrões OGIS	**S**	**S**	**S**	**N**	**S**
Extensível (aceita Plug-Ins)	**S**	**S**	**S**	**N**	**S**
Implementa consultas *ad-hoc*	**N**	**S**	**S**	**N**	**S**
Possui uma versão estável	**S**	**S**	**S**	**S**	**N**
Oferece lista de discussão para desenvolvedores	**S**	**S**	**S**	**N**	**S**
Linguagem orientada a objetos	**S**	**S**	**N**	**S**	**S**
Permite a manipulação dos dados obtidos	**S**	**N**	**S**	**S**	**S**

Quadro 2-2 Comparativo entre os sistemas estudados.

3 Trabalhos Correlatos

Aqui serão apresentadas as principais ferramentas SIG para desktop disponíveis atualmente. Com isso, será possível analisar as vantagens e desvantagens da cada uma e o motivo da escolha da ferramenta JUMP para o desenvolvimento do trabalho.

3.1 Thuban

O Thuban [TH01] é um sistema escrito em *Python* e voltado para a visualização interativa de dados de natureza espacial. Ele possui uma interface amigável e alguns recursos úteis, tais como:

- Suporte a dados vetoriais: ESRI® Shapefiles e conexão PostGIS;
- Suporta dados matriciais: geoTIFF;
- Permite análises (*queries*) e junções de tabelas;
- Possui suporte a projeções;
- Ferramenta de impressão e exportação de vetores;
- API para extensões (plug-ins)
- Suporte multilíngüe

Assim como o JUMP, este sistema é facilmente extensível através de *plugins*. Da mesma forma que Java, Python é uma linguagem orientada a objetos, permitindo maior facilidade na manutenção e expansão (reutilização de classes) do código fonte. A maior desvantagem encontrada no Thuban em relação ao JUMP para os fins deste trabalho é que o Thuban é um sistema voltado para a **visualização** dos dados. Isto o torna, de certa forma, limitado pois não é possível a manipulação dos dados ou mesmo interações bilaterais com o SGBD. O Quadro 3-1 traz informações sobre o site mantenedor do Thuban, a linguagem em que foi programado, o seu tipo de licença e os padrões OGC (OpenGIS Consortium) seguidos pelo mesmo.

Quadro Resumido	
Site Principal:	http://thuban.intevation.org
Mantenedor (responsável):	Intevation GmbH (thuban@intevation.de)
Linguagem (código-fonte):	Python
Licença:	GPL
Padrões OGC:	SFSSQL (PostGIS)

Quadro 3-1 Quadro resumido – Thuban (adaptado de [GEO01])

3.2 Grass GIS

Geographic Resources Analysis Support System, ou GRASS [GR01], é o mais antigo sistema livre para aplicações de SIG. Durante sua trajetória, o sistema incorporou poderosos recursos para área de Geotecnologias tais como:

- Tratamento sobre arquivos matriciais (incluindo recursos para vetorização, análise de correlação/covariância, reamostragem, ajuste das tabelas de cores e geração de superfície através de linhas vetoriais);
- Análises 3D sobre arquivos matriciais (tais como importação de dados 3D – ASCII formato xyz, interpolação e visualização);
- Análises vetoriais (geração de contornos a partir de superfícies matriciais, ferramentas de digitalização, dentre outras);
- Análises de malhas de pontos (triangulação - Delaunay, interpolação para geração de superfície e análises geodésicas);
- Processamento de imagens (composição de cores, ajustes de histograma, ortoretificação, reamostragem, conversão de cores: IHS/RGB);
- Visualização (análises sobre superfícies 3D, camadas vetoriais e camadas matriciais);
- Criação de mapas (postscript e html).

Apesar da extensa lista de funcionalidades, o sistema carece de uma interface gráfica amigável e de uma boa engenharia de software. Ao contrário do Thuban e do JUMP nos quais todo o trabalho do usuário está baseado num ambiente gráfico, o usuário do GRASS precisa recorrer à linha de comando para ter acesso a alguns recursos. Além disso, por não possuir um código orientado a objetos, este sistema dificulta a vida dos desenvolvedores que irão manter e/ou expandir o código do mesmo. Desta forma, se uma instituição pública contratar uma solução baseada no GRASS é bem provável que crie um vínculo de dependência com a empresa contratada. O Quadro 3-2 traz informações sobre o site mantenedor do GRASS, a linguagem em que foi

programado, o seu tipo de licença e os padrões OGC (OpenGIS Consortium) seguidos pelo mesmo.

Quadro Resumido	
Site Principal:	http://grass.itc.it/index.html
Site Secundário (espelho):	http://grass.ibiblio.org/index.html http://www.geog.uni-hannover.de/grass/index.html
Mantenedor (responsável):	Equipe de 19 desenvolvedores (*Core Team*)
Linguagem (código-fonte):	C
Licença:	GPL
Padrões OGC:	SFSSQL (conexão ao PostGIS feita através do PostGRASS)

Quadro 3-2 Quadro resumido – GRASS (adaptado de [GEO01])

3.3 Terraview

Este sistema é um visualizador de bases cartográficas voltado para aplicações de SIG. Ele possui uma interface gráfica com o usuário e capacidade de manipular dados vetoriais (pontos, linhas e polígonos) e matriciais (grades e imagens). Foi desenvolvido pelo Instituto Nacional de Pesquisas Espaciais (**INPE**) utilizando a biblioteca TerraLib. O Terraview [TR01] possui algumas limitações que dificultam o emprego do mesmo em ambientes corporativos:

- Apesar de trabalhar com o PostgreSQL, o TerraView não segue a especificação SFSSQL (OGIS), trabalhando com uma estrutura de dados própria. Isto significa que uma base de dados criada pelo TerraView no PostgreSQL não pode ser acessada pelas aplicações que seguem a SFSSQL (JUMP, Thuban, GRASS, QGIS);
- O projeto não incentiva a criação de uma comunidade para ajudar na manutenção e no desenvolvimento do sistema, tendo como

conseqüência, um site com pouca informação e manuais com abordagens superficiais.

O Quadro 3-3 traz informações sobre o site mantenedor do TerraView, a linguagem em que foi programado, o seu tipo de licença e os padrões OGC (OpenGIS Consortium) seguidos pelo mesmo.

Quadro Resumido	
Site Principal:	http://www.dpi.inpe.br/terraview/index.html
Mantenedor (responsável):	INPE
Linguagem (código-fonte):	C++
Licença:	GPL
Padrões OGC:	Nenhum

Quadro 3-3 Quadro resumido – TerraView (adaptado de [GEO01])

3.4 Quantum GIS (QGIS)

O QGIS [QG01] é um visualizador de dados geográficos que possui poucos recursos para tratamento dos dados, mas permite o acesso a uma grande variedade de dados vetoriais. Mas ele também suporta vários formatos matriciais (ESR® ArcGrid, ERDAS, geoTIFF, etc.). Além disso, com uma crescente comunidade, este projeto também contempla o padrão SFSSQL (OGIS) e já prevê, nas próximas versões, o desenvolvimento de ferramentas para edição de arquivos ESR® Shapefiles e camadas do PostGIS. Apesar de todas as vantagens listadas, o QGIS ainda pode ser considerado apenas como uma grande promessa, pois se encontra ainda em pleno desenvolvimento, estando atualmente na versão 0.6. O Quadro 3-4 traz informações sobre o site mantenedor do QGIS, a linguagem em que foi programado, o seu tipo de licença e os padrões OGC (OpenGIS Consortium) seguidos pelo mesmo.

Quadro Resumido	
Site Principal:	http://qgis.org
Site de Desenvolvimento:	http://sourceforge.net/projects/qgis
Mantenedor (responsável):	Gary Sherman
Linguagem (código-fonte):	C++
Licença:	GPL
Padrões OGC:	SFSSQL (PostGIS)

Quadro 3-4 Quadro resumido – QGIS (adaptado de [GEO01])

4 Implementação

Este capítulo descreve o resultado prático do trabalho: A implementação de uma extensão que permite aos usuários do JUMP realizar consultas SQL *ad hoc* em bancos de dados PostgreSQL.

4.1 PostGISQueryPlugIn

Um ***PlugIn*** no JUMP é um objeto que executa uma ação qualquer, em resposta a uma seleção de menu ou o pressionamento de um botão. Um dos vários *PlugIns* para o JUMP disponíveis é o **PostGisDriver,** desenvolvido pela **Refractions Research** [RR01]. Tal *PlugIn* permite que sejam acessados os dados provenientes de um base de dados do PostgreSQL contendo extensões espaciais e que os mesmos possam ser exibidos e manipulados dentro do JUMP.

O principal problema com o PostGisDriver é que não são permitidas consultas *ad-hoc*. O usuário seleciona a base de dados e em seguida qual tabela deseja visualizar. Isso torna o mesmo muito limitado, pois o usuário não tem a liberdade de realizar consultas que retornem exatamente o que ele necessita. Por exemplo, poderíamos querer descobrir “quais os rios que cruzam as estradas em cidades americanas com mais de cem mil habitantes”. Isso não seria possível com a abordagem original do mesmo. Por este motivo, tornou-se clara a necessidade da extensão das suas capacidades, vista a facilidade concedida pelo JUMP para alterações desta natureza. A partir desta necessidade, foi feito um estudo para entender o funcionamento do JUMP, do *PlugIn* existente e das possíveis alterações para permitir fazer-se as consultas da maneira necessária. O resultado obtido foi o **PostGISQueryPlugIn** que estende as capacidades originais do PostGisDriver, permitindo com isso que o usuário realize as suas pesquisas de forma mais precisa e livre.

A manipulação e análise dos dados continuam sendo feitas através da própria interface do JUMP. Isso garante que o software continua como o original de forma que as alterações realizadas refletem apenas na aquisição dos dados. Isso só foi possível devido à capacidade do JUMP de ser estendido através de *PlugIns*, sem a necessidade de alteração direta de seu código fonte.

4.2 Arquitetura e Funcionamento

4.2.1 Arquitetura do PostGISQueryPlugIn

A arquitetura do **PostGISQueryPlugIn** é baseada no PostGISDriver [PGD01] desenvolvido pela **Refractions Research** [RR01]. Podemos observar uma ilustração desta aquitetura na Figura 4-1. Resumidamente, a classe PostGISQueryExtension é carregada pelo Gerenciador de PugIns do JUMP e na seqüência, através do método *initialize*, é inicializada a classe PostGISQueryPlugIn.

As classes PostGISQueryLoadDataSourceQueryChooser, PostGISQueryDataSource, PostGISQueryDataSourceQueryChooser e PostGISQueryLoadDriverPanel servem basicamente para implementar a interface gráfica do *PlugIn* com o usuário, receber os dados de conexão informados pelo usuário, e, guardá-los para sua posterior utilização.

A classe PostGISQueryConnection é a classe responsável por abrir a conexão com o banco de dados e fazer uma "tradução" da consulta informada pelo usuário para um formato que pode ser entendido pelo JUMP. A seguir, cada uma das classes constituintes do *PlugIn* serão detalhadas e seus diagramas UML (Unified Modeling Language) poderão ser observados.

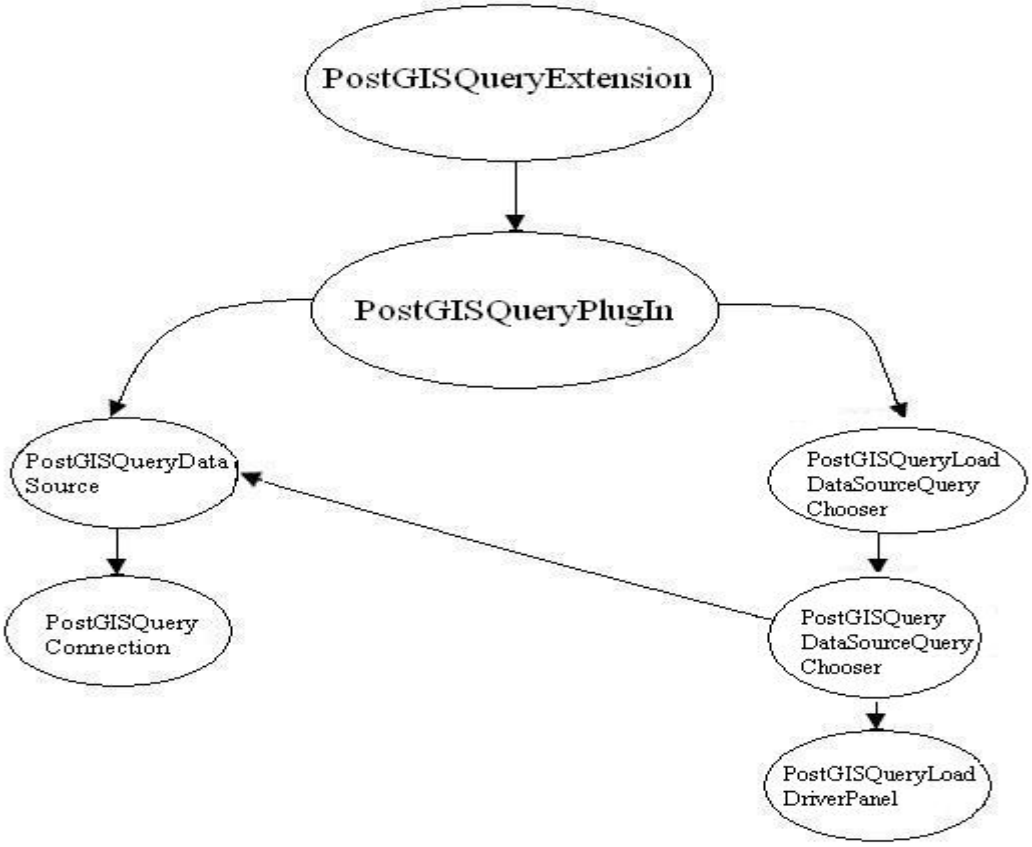

Figura 4-1 Arquitetura do PostGISQueryPlugIn

4.2.2 Classe PostGISQueryExtension

Uma extensão é uma coleção de classes e recursos de suporte que provêem funcionalidades adicionais ao JUMP. Extensões são “empacotadas” em arquivos JAR. Para o usuário, estender o JUMP é tão simples como copiar um arquivo de extensão JAR para o diretório de *PlugIns* do mesmo. O *Workbench* do JUMP irá procurar dentro do arquivo JAR por subclasses de *Extension*.

Seguindo a sua classe ancestral, a classe **PostGISQueryExtension** executa o método *Configure(PlugInContext context)* que neste caso, simplesmente chama o método *initialize(PlugInContext context)* da classe *PostGISQueryPlugIn* o qual será explicado mais adiante. Um diagrama UML da classe *PostGISQueryExtension* pode ser observado na figura 4-2.

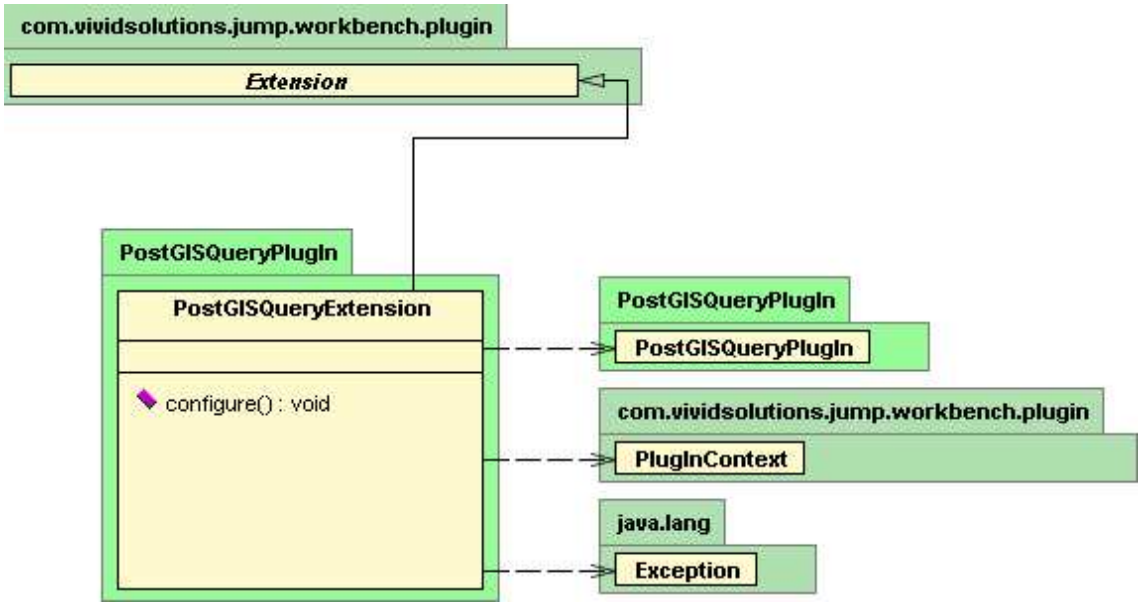

Figura 4-2 Diagrama UML da classe PostGISQueryExtension

4.2.3 Classe PostGISQueryPlugIn

Como dito anteriormente, um *PlugIn* é um objeto que executa uma simples ação, em resposta a uma seleção de menu ou o pressionamento de um botão. Todos os itens de menu do JUMP são na verdade *PlugIns* e são lidos pela classe, sendo esta a *JUMPConfiguration*. Um *PlugIn* possui três métodos: *#initialize, #execute* e *#getName*. O método *initialize* é chamado quando o *WorkBench* é iniciado. *#execute* é chamado quando o *PlugIn* é acionado, por exemplo, quando o usuário seleciona um item de menu ou clica em um botão da barra de ferramentas. *#getName* retorna uma breve descrição do *PlugIn* em questão, por exemplo, para dar um nome ao item de menu.

O método *initialize* foi implementado de modo a criar um objeto do tipo *PostGISQueryDataSource* e, em seguida, passar o mesmo como parâmetro para o construtor da classe *PostGISQueryLoadDataSourceQueryChooser* criando então um objeto da mesma, que será adicionado ao *Workbench* do JUMP através do método *addLoadDataSourceQueryChooser* implementado na classe *DataSourceQueryChooserManager*. O usuário então, poderá acessar a funcionalidade a partir do item de menu *File -> Load Dataset(s)*. O trecho de código da função *initialize* pode ser visto abaixo:

```
public void initialize(PlugInContext context) throws Exception {
    PostGISQueryDataSource dataSource = new PostGISQueryDataSource();
    PostGISQueryLoadDataSourceQueryChooser loadChooser = new
      PostGISQueryLoadDataSourceQueryChooser(dataSource);

    DataSourceQueryChooserManager.get(
      context.getWorkbenchContext().getWorkbench().getBlackboard()
    ).addLoadDataSourceQueryChooser(loadChooser);
  }
```

A função da classe *DataSourceQueryChooserManager* é a de registrar *DataSourceQueryChoosers* na inicialização do JUMP Workbench, como é o caso do *PostGISQueryDataSourceQueryChooser*. Um diagrama UML da classe *PostGISQueryPlugin* pode ser observado na figura 4-3.

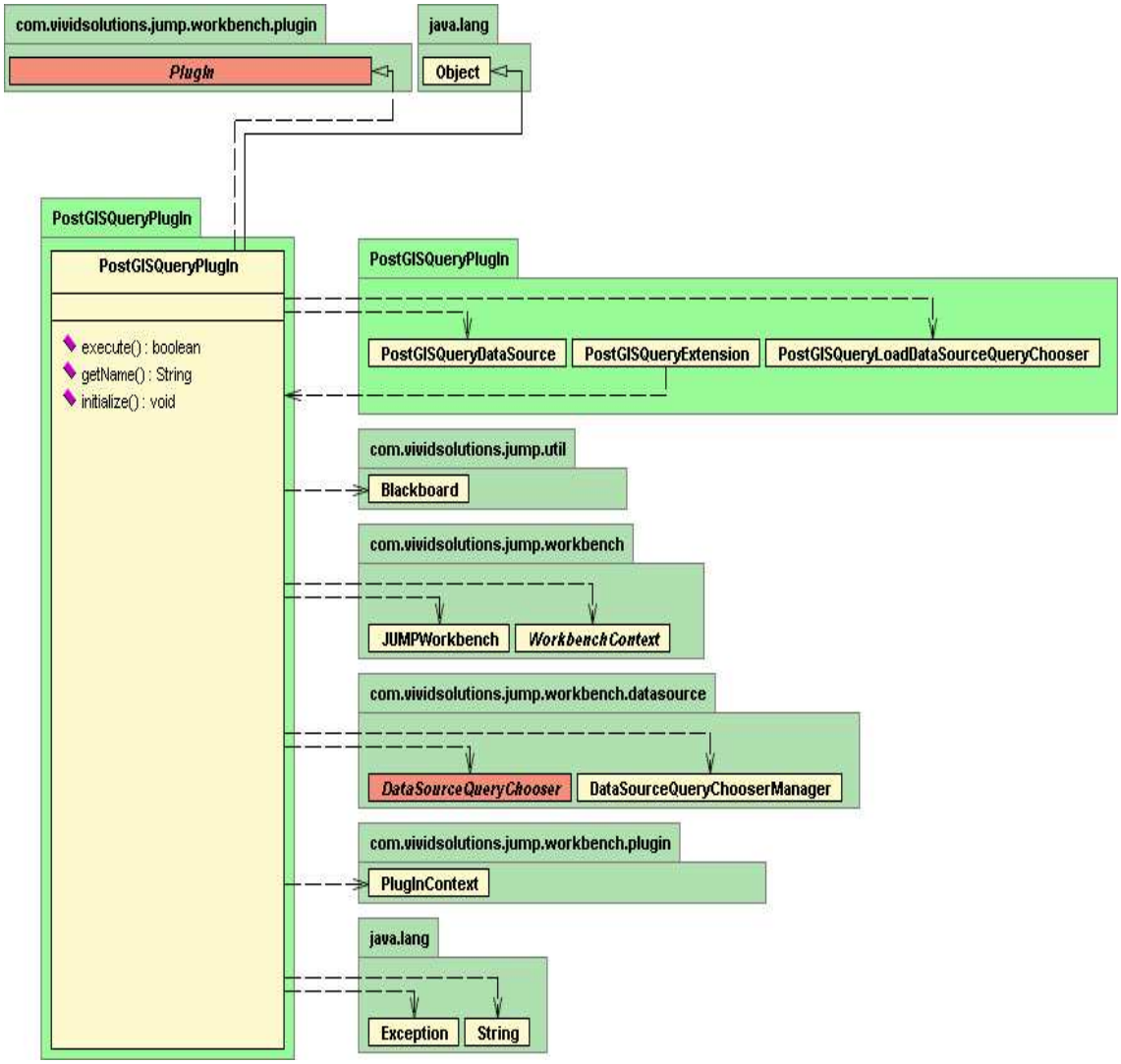

Figura 4-3 Diagrama UML da classe PostGISQueryPlugIn

4.2.4 Classe PostGISQueryDataSource

A classe *PostGISQueryDataSource* estende a classe *DataSource* do JUMP. Sua principal função é criar uma conexão com o banco de dados através do método *getConnection* que pode ser observado na listagem abaixo. Além disso, nesta classe são definidas algumas variáveis que guardam valores de configuração da conexão juntamente com os dados informados pelo usuário do JUMP. Um diagrama UML da classe *PostGISQueryDataSource* pode ser observado na figura 4-4.

```
public Connection getConnection() {
  if (conn == null) conn = new PostGISQueryConnection();
  conn.setProperties(getProperties());
  return(conn);
}
```

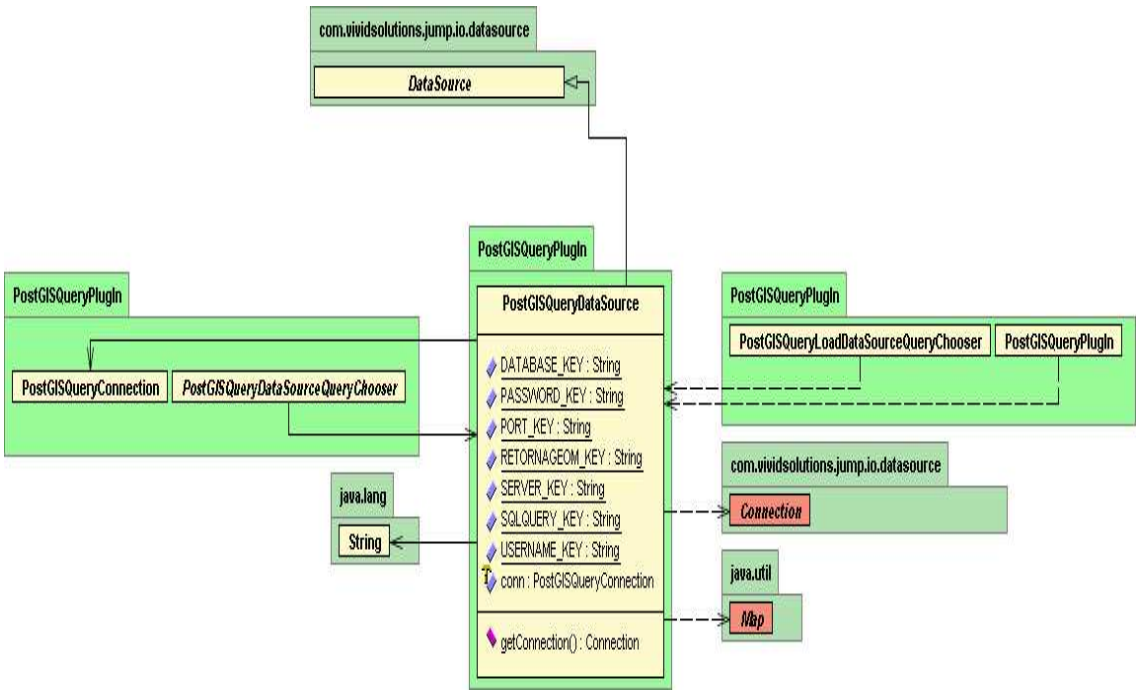

Figura 4-4 Diagrama UML da classe PostGISQueryDataSource

4.2.5 Classe PostGISQueryDataSourceQueryChooser

Classe que implementa uma DataSourceQueryChooser, é a camada intermediária entre a interface gráfica do *PlugIn* e o JUMP. É responsável pela validação dos dados informados pelo usuário e por um guardar os mesmos em uma estrutura do tipo *HashMap* de JAVA. Um diagrama UML da classe *PostGISQueryDataSourceQueryChooser* pode ser observado na figura 4-5.

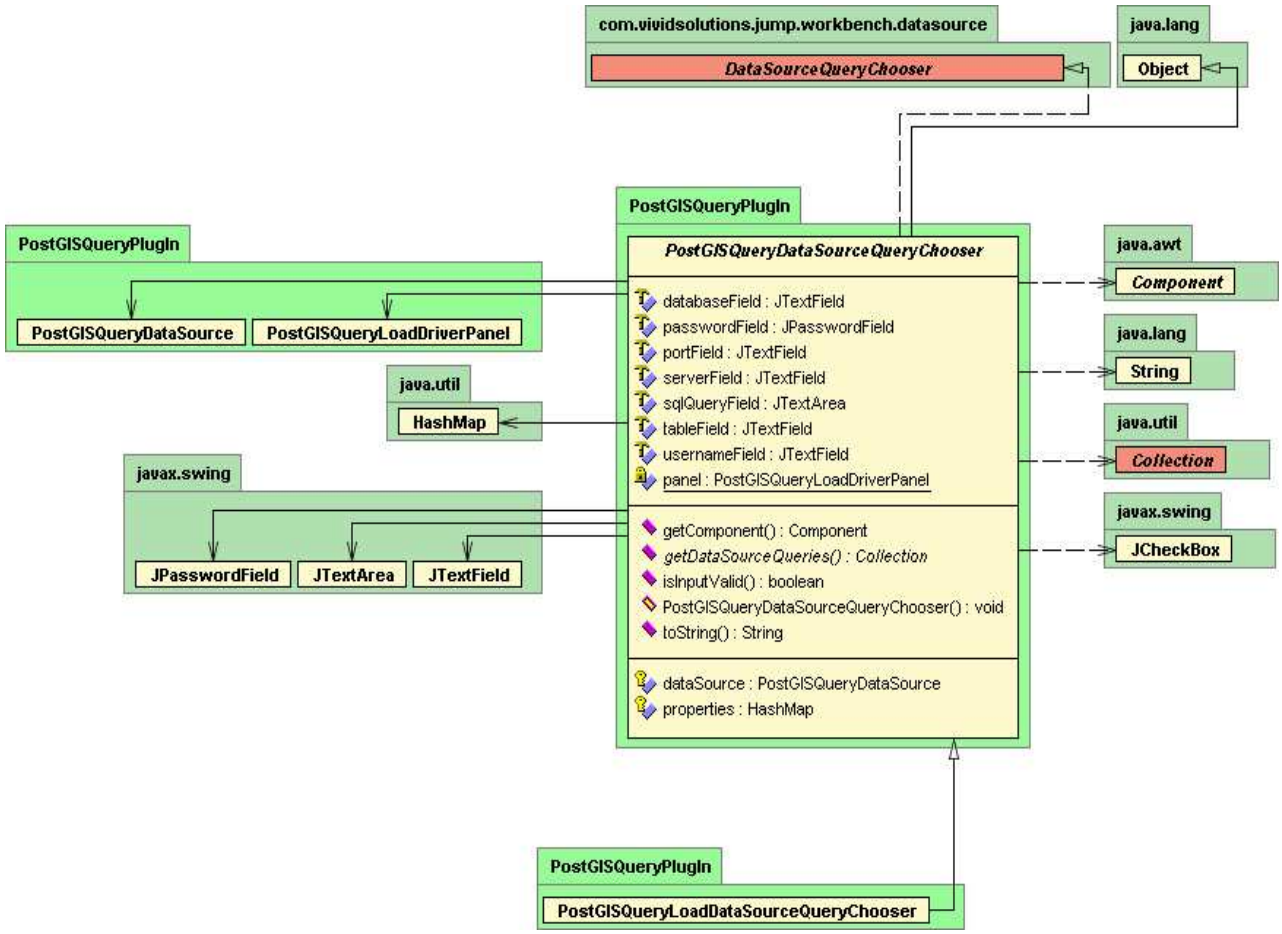

Figura 4-5 Diagrama UML da classe PostGISQueryDataSourceQueryChooser

4.2.6 Classe PostGISQueryLoadDataSourceQueryChooser

Esta classe estende a classe PostGISQueryDataSourceQueryChooser e também implementa DataSourceQueryChooser do JUMP. Cria um objeto do tipo PostGISDataSourceQuery (nativo do PostGISDriver) através do método getDataSourceQueries (esta operação pode ser observada na listagem abaixo). Tal consulta é então passada para o PostGISQueryConnection que fará o seu tratamento e envio ao Banco de Dados. Um diagrama UML da classe *PostGISQueryLoadDataSourceQueryChooser* pode ser observado na figura 4-6.

```
public Collection getDataSourceQueries() {
  StringBuffer sql = new StringBuffer();
  Map properties = super.getProperties();
sql.append((String)properties.get(PostGISQueryDataSource.SQLQUERY_KEY));
  PostGISDataSourceQuery query = new PostGISDataSourceQuery(
    getDataSource(), sql.toString(), "PostGISQuery");
  query.setProperties(getProperties());
  List queries = new ArrayList();
  queries.add(query);
  return(queries);
}
```

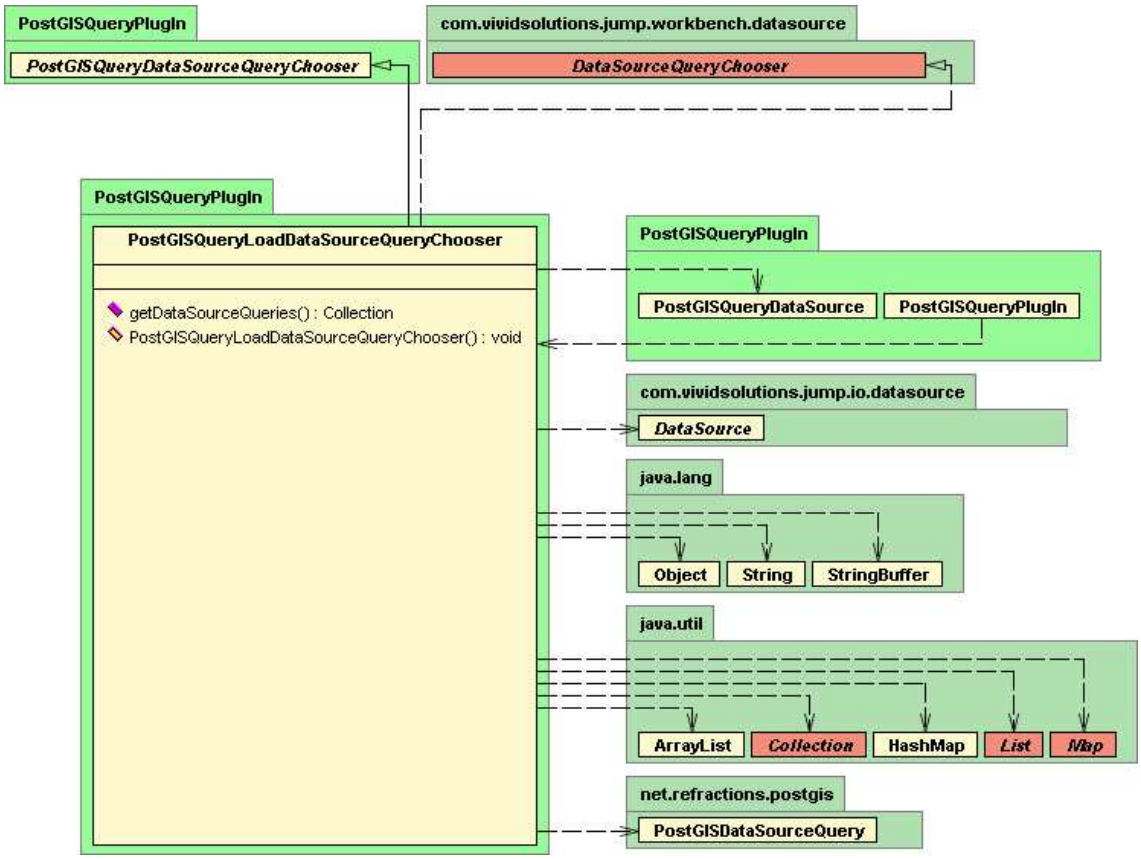

Figura 4-6 Diagrama UML PostGISQueryLoadDataSourceQueryChooser

4.2.7 Classe PostGISQueryLoadDriverPanel

Esta classe implementa *JPanel* e é responsável pelo layout da interface gráfica do *PlugIn* e por transferir os dados informados para as classes inferiores da arquitetura. Um diagrama UML da classe *PostGISQueryLoadDriverPanel* pode ser observado na figura 4-7.

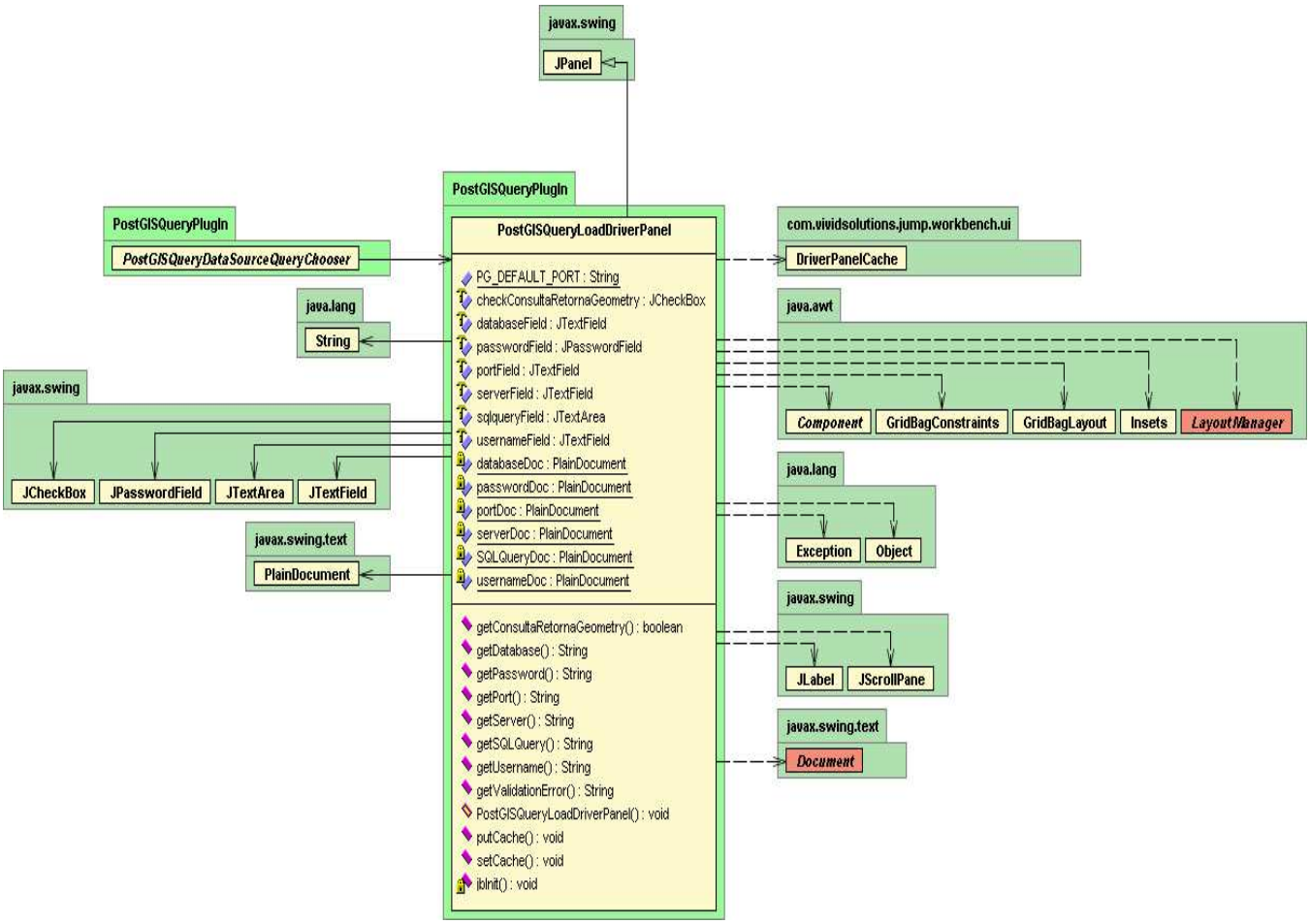

Figura 4-7 Diagrama UML da classe PostGISQueryLoadDriverPanel

4.2.8 Classe PostGISQueryConnection

Trata-se da classe mais importante do *PlugIn* desenvolvido, implementando a classe *Connection* do JUMP. É responsável por abrir a conexão com o banco de dados PostgreSQL através do método *connect* (observar a listagem abaixo), e principalmente, faz a "tradução" da consulta informada pelo usuário para o modelo aceito pelo JUMP. O método consiste em executar a consulta no banco de dados e, em seguida, utilizando a classe ResultSetMetaData de Java, verificar os tipos de cada uma das colunas de retorno para então preencher um objeto *Well-Known Text* (WKT) [WKT01] com os tipos esperados para que dessa maneira o JUMP saiba quais atributos devem ser desenhados na tela. Um diagrama UML da classe *PostGISQueryConnection* pode ser observado na figura 4-8.

```
private java.sql.Connection connect() {
  try {
    String jdbcUrl = "jdbc:postgresql://" + server + ":" + port + "/" + database;
    Class.forName( "org.postgresql.Driver" );
    return(DriverManager.getConnection( jdbcUrl, username, password ));
  }
  catch(ClassNotFoundException cnfe) {
    throw new IllegalStateException("Não foi possível ler o driver PostgresSQL: "
+ cnfe.getMessage());
  }
```

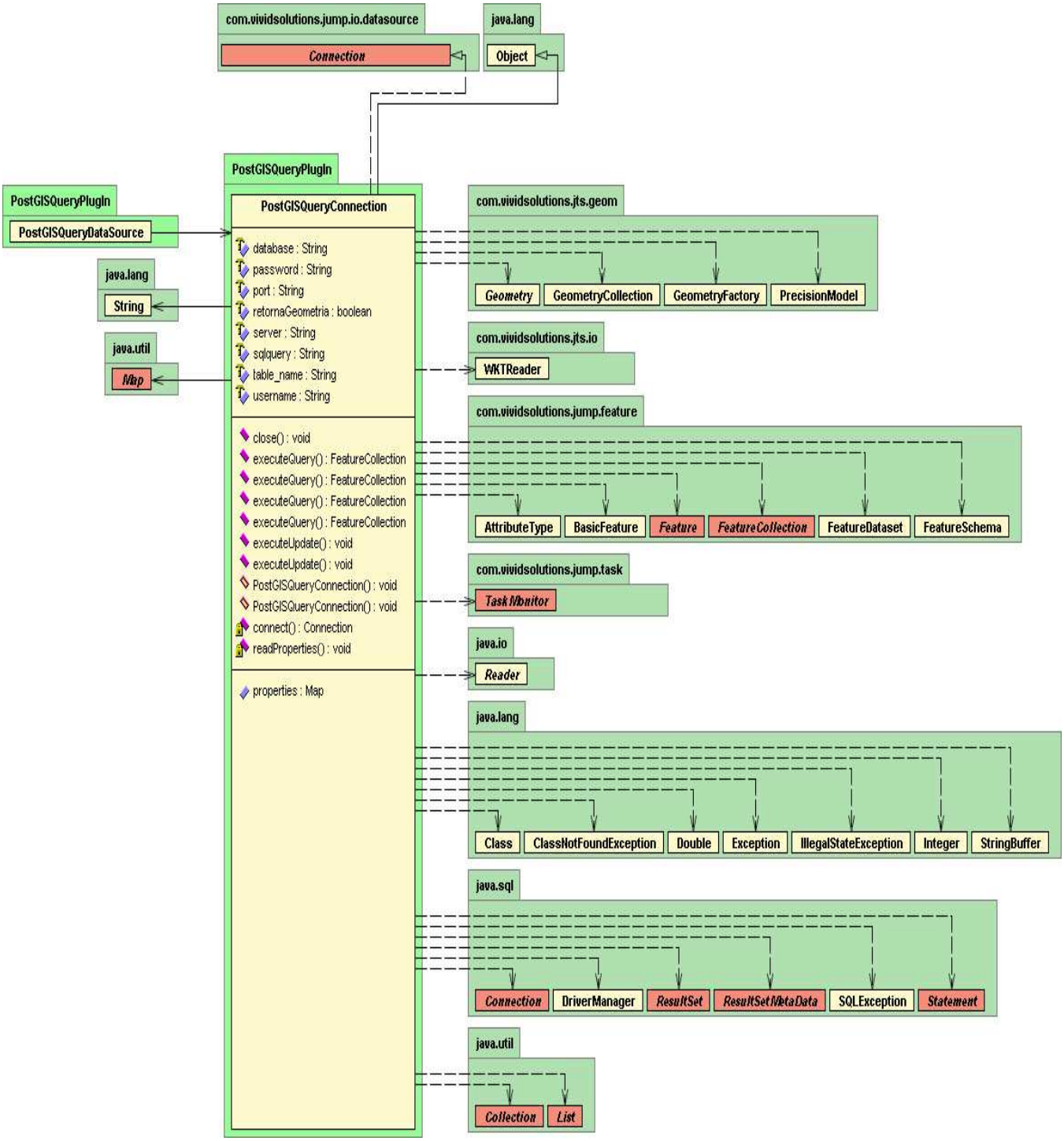

Figura 4-8 Diagrama UML da classe PostGISQueryConnection

4.3 Dificuldades de implementação

Alguns problemas ocorreram durante a implementação do **PostGISQueryPlugIn.** Entre eles podemos citar o pouco conhecimento em algumas das tecnologias envolvidas como foi o caso do PostGreSQL e do próprio JUMP. Mas, a maior de todas as dificuldades na implementação do *PlugIn* foi o fato de termos de realizar uma "conversão" da instrução SQL informada pelo usuário para uma instrução em WKT que é uma forma de trocar dados de geometrias em formato ASCII contida nas especificações do OGIS [OG01].

O processo consiste em executarmos através do **JDBC** [JDBC01] a consulta informada pelo usuário e obtermos um *ResultSet* e, em seguida, extraímos os meta-dados deste *ResultSet*. Após isso, reconstruímos a consulta inserindo a função *AsText* do PostGreSQL em todos os itens constituintes da cláusula SELECT da consulta que forem do tipo **geometry** e, finalmente, definimos os atributos WKT que serão usados pelo JUMP para retornar as informações geométricas ao usuário.

Para consultas simples do tipo "SELECT * FROM CITIES" o processo funciona perfeitamente (originalmente a implementação era assim), mas para consultas que envolvem mais de uma tabela (JOINS) esse esquema se torna inviável, pois precisamos utilizar *ALIAS* nas nossas consultas. Acontece que na primeira vez que enviamos a consulta ao banco para podermos extrair os meta-dados, o mesmo retorna os nomes das colunas já como o *ALIAS* que indicamos. Isso não permite que seja utilizada a função *AsText* do PostGreSQL e, conseqüentemente, não teremos as colunas do tipo **geometry** reconhecidas pelo JUMP.

A solução ideal para este problema seria a implementação de um método que, utilizando expressões regulares, pudesse “decompor” a *query* informada para que as colunas informadas pudessem ter seus tipos validados um a um. Infelizmente, o tempo reservado para o desenvolvimento deste trabalho não seria suficiente para a implementação de tal método. Logo, se tornou necessário ignorar a parte do processo que realiza o reconhecimento dos tipos indicados no *ResultSet*. Dessa maneira, o usuário deve informar a consulta já com a função *AsText* o que de certa forma constitui uma limitação do *PlugIn*. Outra conseqüência é que o primeiro item da cláusula SELECT da consulta deve ser do tipo **geometry**, a não ser que o usuário informe explicitamente que a consulta não retorna dados geométricos, através de um *checkBox* para esta função.

Com essa abordagem, é possível executar qualquer consulta SQL suportada pelo PostgreSQL desde que, como indicado, a primeira coluna da cláusula SELECT seja do tipo **geometry** e esteja dentro da função *AsText*. Caso o usuário queira executar uma consulta sem a obtenção dos dados espaciais, deverá desmarcar o *chekBox* “A consulta retorna dados espaciais”. Observando esta limitação, pode-se chegar à conclusão de que a mesma não deve atrapalhar o uso do *PlugIn*, já que o usuário do PostgreSQL teria de toda maneira que utilizar a função *AsText* em suas consultas. Naturalmente, no futuro, o ideal seria a implementação do método de “decomposição” das consultas descrito acima.

4.4 Resultados obtidos

Com a implementação do PostGISQueryPlugIn, é possível a realização de pesquisas rebuscadas e de acordo com as necessidades específicas do usuário. Tais pesquisas podem conter todas as funções dos PostGIS aceitas pelo JUMP e podem ser de qualquer complexidade. A única restrição do *PlugIn* é o fato da primeira coluna da cláusula SELECT da consulta ser obrigatoriamente do tipo *geometry* e estar dentro da função *AsText* do JUMP, isso caso no checkBox "Consulta retorna tipo Geometry" estiver marcado.

O *PlugIn* cumpre a proposta de não alterar as características nativas do JUMP comportando-se apenas como uma mera extensão que agrega uma nova funcionalidade ao mesmo.

Algumas das consultas que podem ser realizadas com a ferramenta, utilizando-se o produto deste trabalho, são apresentadas nas figuras, 4-9, 4-10, 4-11 e 4-12.

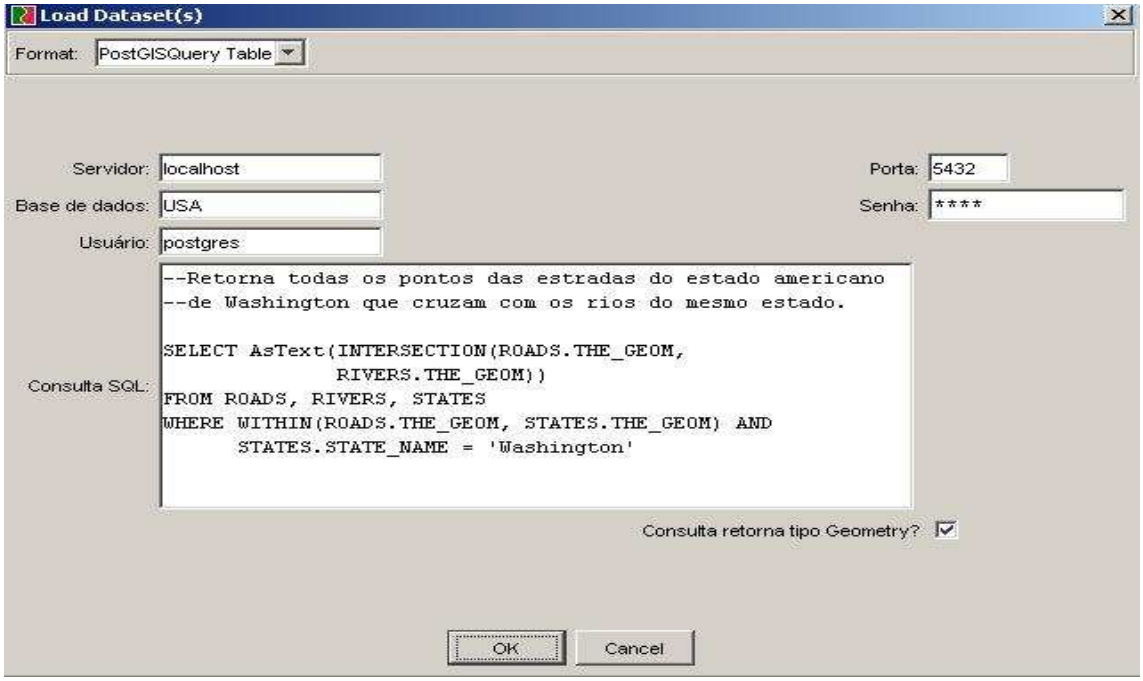

Figura 4-9 Tela do PlugIn responsável por receber as informações da consulta.

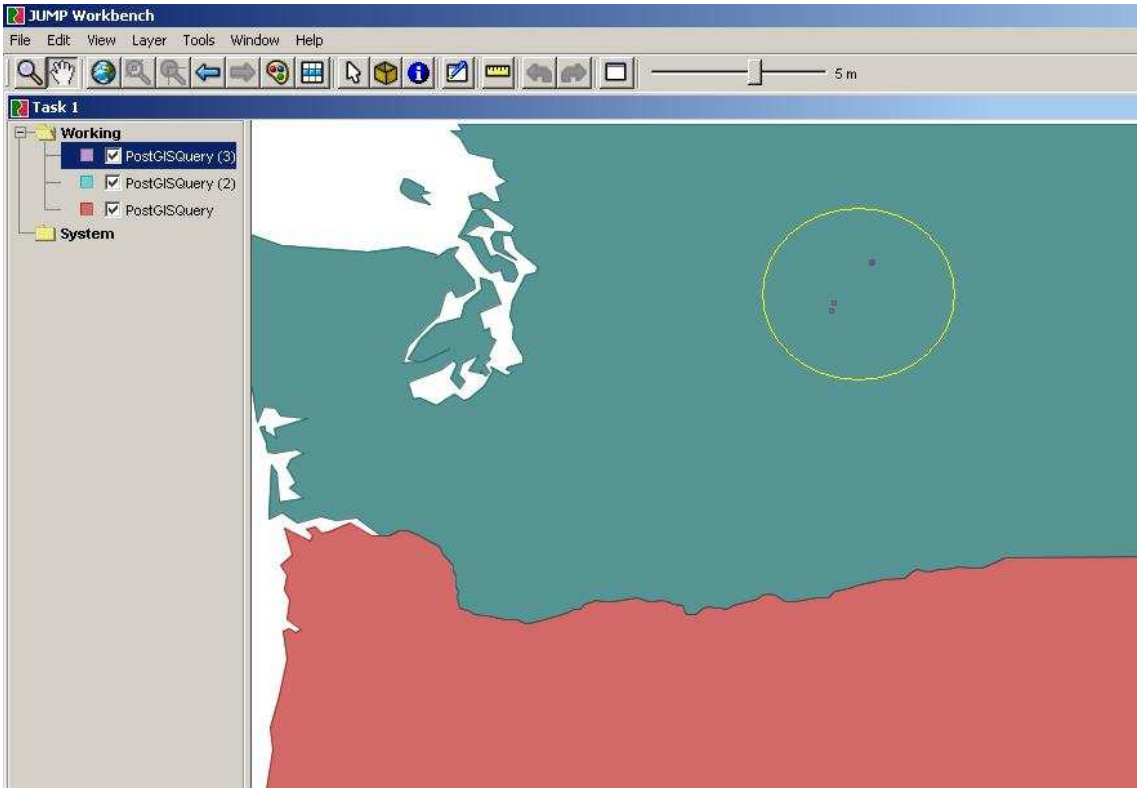

Figura 4-10 Resultado da consulta da Figura 4-9

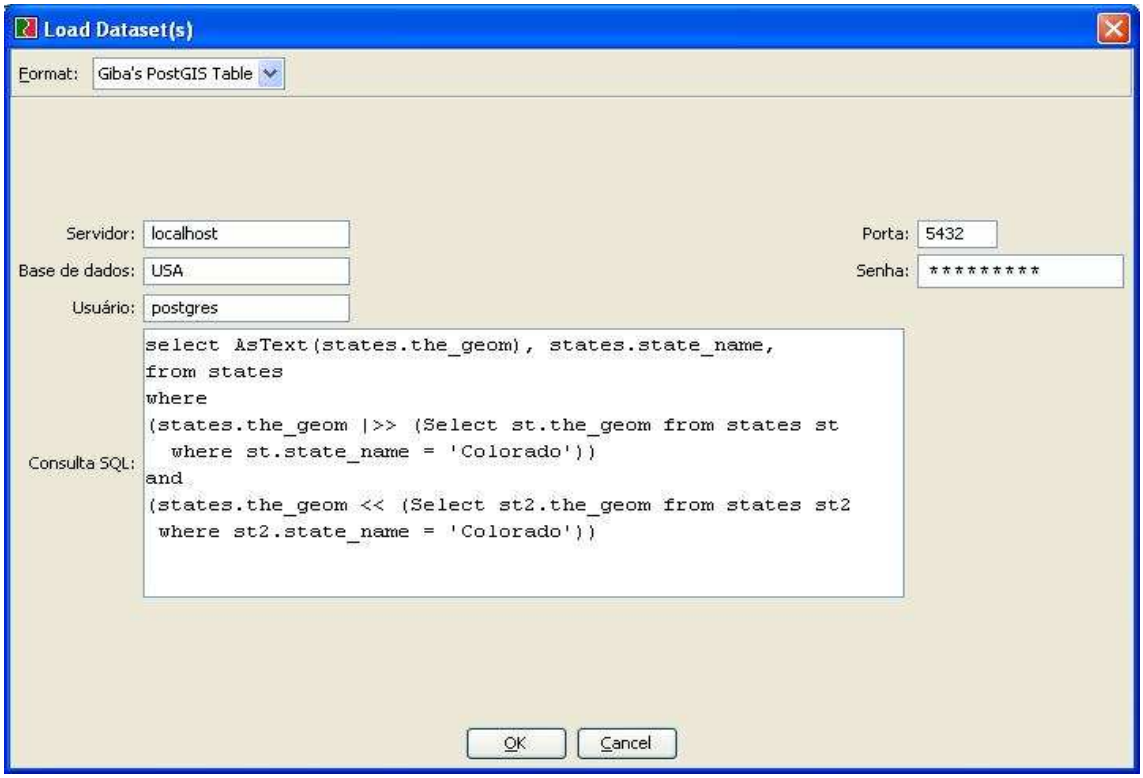

Figura 4-11 Consulta que exibe os estado à noroeste do Colorado

Figura 4-12 Resultado da consulta da Figura 4-11

4.5 Aplicação

A principal aplicação do *PlugIn* será dentro do projeto GOLAPA [FTS01, GHP03] do Centro de Informática da UFPE, servindo como uma grande melhoria no JUMP que é utilizado para a realização das consultas na parte geográfica do mesmo. A flexibilidade obtida com essa extensão será de grande utilidade para os pesquisadores do projeto, pois agora as consultas necessárias para seus estudos podem ser realizadas de forma mais específica, descartando o grande problema de não poder realizar consultas *ad hoc* com o *PostGISDriver*.

Além disso, está sendo analisada a possibilidade da publicação permanente do *PlugIn* no projeto do JUMP, sendo que a manutenção do mesmo ficaria a cargo do autor deste trabalho. Com isso, o mesmo poderia ser difundido entre a comunidade usuária do JUMP, além de provavelmente ser melhorado pela mesma.

4.6 Exemplos

Alguns exemplos de consultas que podem ser feitas com o JUMP utilizando o PostGISQueryPlugIn:

a) **Operação AsText**

```
//Selecionar a geometria do estado do Colorado.
   SELECT AsText(THE_GEOM)
   FROM STATES
   WHERE STATES.STATE_ABBR = 'CO'
```

b) **Operador Distance**

```
//Selecionar todas as cidades que estão a uma distância de 200KM da
//cidade de Havre no estado de Montana.
   SELECT AsText(CITIES.THE_GEOM),
                CITIES.CITY_NAME,
                CITIES.STATE_NAME
   FROM CITIES
   WHERE (DISTANCE(GeometryFromText('POINT(-109.67985528815
                48.5438182640186)',-1), CITIES.THE_GEOM)*100) >0 AND
                (DISTANCE(GeometryFromText('POINT(-109.67985528815
                  48.5438182640186)',-1), CITIES.THE_GEOM)*100) <= 200
```

c) **Operador Within**

```
//Selecionar todas as cidades  que estão dentro da geometria do
//estado do Colorado 'CO'.
   SELECT AsText(CITIES.THE_GEOM),
                CITIES.CITY_NAME,
                CITIES.STATE_NAME
   FROM CITIES, STATES
   WHERE WITHIN(CITIES.THE_GEOM, STATES.THE_GEOM) AND
              STATES.STATE_ABBR = 'CO'
```

d) **Operador Intersects**

```
//Selecionar todas as intersecções entre estradas e rios.
    SELECT AsText(INTERSECTION(ROADS.THE_GEOM,
                RIVERS.THE_GEOM))
     FROM ROADS, RIVERS
```

e) **Operações de buffer**

```
//Selecionar todas as cidades dentro do buffer gerado a partir do estado do
//Colorado
    SELECT AsText(CITIES.THE_GEOM),
                CITIES.CITY_NAME,
                CITIES.STATE_NAME
    FROM CITIES
    WHERE WITHIN(CITIES.THE_GEOM,
                    BUFFER((SELECT STATES.THE_GEOM
                                FROM STATES
                                WHERE STATES.STATE_ABBR =
                                    'CO'),2,8))
```

f) **Testes com Operadores Posicionais**

```
//Selecionar os estados posicionados à Noroeste do Estado do Colorado

    select AsText(states.the_geom), states.state_name,
                    states.state_abbr
    from states
    where (states.the_geom |>> (Select st.the_geom from states st
            where st.state_name = 'Colorado')
            and (states.the_geom << (Select st2.the_geom from
            states st2 where st2.state_name = 'Colorado')))
```

5 Conclusões e trabalhos futuros

Este capítulo tem como objetivo apresentar algumas considerações finais sobre os principais tópicos abordados neste trabalho, incluindo as contribuições alcançadas e indicações para trabalhos futuros.

5.1 Conclusões e Considerações Finais

O trabalho realizado teve como objetivo estudar as soluções existentes para análise e visualização de dados de natureza geográfica e permitir que a ferramenta escolhida realize consultas *ad hoc*. Além disso, as características originais do software (o JUMP) não foram alteradas, de modo que todas as necessidades foram solucionadas com a implementação de uma extensão do mesmo.

Com a utilização do *PlugIn,* agora é possível a realização de consultas *ad hoc*, elimidando um grande obstáculo da utilização do JUMP por usuários que desejam realizar as suas pesquisas em bancos de dados com suporte a operadores espaciais como é o caso do PostgreSQL.

Com esta contribuição, os pesquisadores do projeto GOLAPA poderão realizar as suas consultas espaciais de forma mais seletiva, resultando com isso em uma grande economia de tempo. Além disso, o *PlugIn* (resultado deste trabalho) poderá ser disponibilizado para a comunidade usuária do JUMP. Isso fará com que um grande número de pessoas possa utilizá-lo e até mesmo melhorá-lo.

5.2 Trabalhos Futuros

Algumas melhorias podem ser feitas no protótipo do **PostGisQueryPlugIn**. Uma delas é a limitação imposta pelo fato de que, caso o usuário pretenda que a consulta retorne dados espaciais, o mesmo ter que informar como primeiro parâmetro da cláusula SELECT um tipo *geometry* e este estar dentro da função *AsText*. Uma possível solução para a mesma seria a implementação de um método que, utilizando expressões regulares, fosse capaz de "decompor" a consulta informada para que as colunas pudessem ter seus tipos validados um a um. Infelizmente, o tempo reservado para o desenvolvimento deste trabalho não seria suficiente para a implementação de tal método.

Outra funcionalidade que poderia ser resolvida no futuro seria a melhoria no tratamento de exceções que apesar de serem gerenciadas pelo JUMP, ainda exibem erros que não deveriam ser mostrados ao usuário final. Além disso, é necessário um bom tratamento na interface gráfica a fim de melhorar a aparência estética da mesma e a tradução de seus textos para Inglês, pelo fato de ser a língua oficial do JUMP e para que os outros desenvolvedores envolvidos no projeto possam contribuir com maior facilidade.

Referências Bibliográficas

[Cla97] Clarke, Keith C., Getting Started with Geographic Information Systems, Prentice Hall, USA, 1997.

[CCH+96] G. Câmara, M. A. Casanova, A. S. Hemerly, G. C. Magalhães, C. M. B. Medeiros. Anatomia de Sistemas de Informação Geográfica. 10ª Escola de Computação, 1996. http://www.dpi.inpe.br/geopro/livros/anatomia.pdf (último acesso em agosto de 2005)

[Dem97] Michael N. Demers. Fundamentals of Geographic Information Systems, Wiley, USA, 1997.

[Fid03] Fidalgo, Robson do Nascimento, Tese de Doutorado - Integração de Processamento Analítico com Sistemas de Informações Geográficas para Suporte à Decisão em Ambientes de Data Warehouse Geográfico – Cin/UFPE, 2005.

[FTS01] R. N. Fidalgo, V. C. Times, F. F. Souza. ***GOLAPA: Uma Arquitetura Aberta e Extensível para Integração entre SIG e OLAP***. In Proc. GeoInfo 2001, http://www.ic.unicamp.br/~cmbm/geoinfo/papers/149robson.pdf, (ultimo acesso em agosto de 2005).

[GEO01] Helton N. Uchoa, Paulo Roberto F. **Geoprocessamento com Software Livre.** Disponível em http://www.geolivre.org.br (último acesso em agosto de 2005)

[GHP03] ***GOLAPA Home Page***, http://www.cin.ufpe.br/~golapa/ (ultimo acesso em agosto de 2005)

[GNU03] ***GNU Gerenal Public License***, http://www.gnu.org/licenses/gpl.html (último acesso em agosto de 2005)

[GR01] **GRASS, Geographic Resources Analysis Support System,** http://grass.itc.it/index.html (ultimo acesso em agosto de 2005)

[GR02] Queiroz, Gilberto Ribeiro de, Proposta de Tese de Mestrado – Extensão do SGBD PostgreSQL com Operadores Espaciais – INPE.

[JDBC01] **JDBC Technology**, http://java.sun.com/products/jdbc/ (último acesso em agosto de 2005).

[JS01] Silva, Joel da, Dissertação de Mestrado – Integrando Serviços Analíticos e Geográficos para Suporte à Decisão na Web – Cin/UFPE, 2004.

[MGR91] David J. Maguire, Michael F. Goodchild, David W. Rhind., Geographical Information Systems Principles and Applications, Volume 1, Longman, 1991.

[OGC01] ***Open GIS Consortium Inc.***, http://www.opengis.org (último acesso em agosto de 2005).

[PG01] ***PostGreSQL.***, http://www.postgres.org (último acesso em agosto de 2005).

[PGD01] ***JUMP PostGIS Driver 1.1.0***, http://www.jump-project.org/ (último acesso em agosto de 2005).

[QG01] **QGIS**, http://qgis.org, ([ultimo acesso em agosto de 2005)

[RR01] **Refractions Research**, http://www.refractions.net/ (último acesso em agosto de 2005).

[TH01] **Thuban**, http://thuban.intevation.org/ (ultimo acesso em agosto de 2005)

[TR01] **TerraView**, http://www.dpi.inpe.br/terraview/index.html (último acesso em agosto de 2005)

[VIV01] **Vivid Solutions**, http://www.vividsolutions.com/ (último acesso em agosto de 2005)

[Ylu00] Fernandes, Damires Yluska de Souza, GeoVisual Interface – Uma Interface para Consultas Visuais em Bancos de Dados Geográficos, Dissertação de Mestrado, CIn-UFPE, 2000.

DATAS E ASSINATURAS

_____

Gilberto Antonio da Silva Júnior
Graduando

_____

Prof.a Dr.a Valéria Cesário Times
Orientadora

Recife, 29 de Agosto de 2005