Nome: Albano Roberto Drescher Von Maywitz

Semana 2

Tarefa 4

Considerando um SGBD de sua preferência, descreva sobre o suporte deste SGBD para diferentes modelos de dados geográficos, operações e formas de visualização que sejam necessários para o armazenamento e a identificação da localização dos acessos ao SEU sistema web. Indique as extensões e comandos necessários para se obter as regiões geográficas que mais tiveram acesso ao sistema.

Resposta

O Sistema de Gerencimento de Banco de Dados (SGBD) escolhido foi postgreSQL. Será abordado como os dados geográficos são criados e manipulados, bem como, uma breve explicação sobre dados geográficos e espaciais.

Primeiro, vou abordar a diferença entre dados Geográficos e Espaciais:

> *Dado espacial é qualquer tipo de dado que descreve fenômenos aos quais esteja associada alguma dimensão espacial. Dados geográficos ou georreferenciados são dados espaciais em que a dimensão espacial está associada à sua localização na superfície da terra, num determinado instante ou período de tempo [CCHM96].*

Os Sistemas de Informações Geográficas (SIG) são equipamentos e meios tecnológicos para se estudar o espaço terrestre. Ou seja, o banco de dados geográfico é um dos componentes mais importantes para quem trabalha com SIGs. Um erro muito comum para quem o utiliza um SIG é que inicialmente não se preocupam com a organização, armazenamento e normatização dos dados, onde um banco de dados espacial bem estruturado pode solucionar, bem como previnir esses erros.

Uma informação importante para quem está criando um banco de dados espacial é para que ele está sendo criado, pois sem essa resposta pode-se criar um banco com muita informação desnecessária para a aplicação que utilizará o banco, um exemplo, o banco de dados de um estudo médico que quer determinar onde há mais incidência de determinada doença não precisa conter tantas informações quanto o banco de dados do Google Maps, com informações de restaurantes, rodovias e etc.

As informações que são armazenadas em um banco de dados são divididas duas formas, dados Alfanuméricos, que são textos e números, esses tipos de dados são utilizados em um banco de dados relacional comum, também existe a forma de dados espaciais que podem representar informações sobre o local físico e a forma de objetos geométricos. Esses objetos podem ser locais como cidades, edifícios, representados por pontos ou objetos mais complexos como países, estradas ou lagos, representados por múltiplos polígonos, conjuntos de linhas ou polígonos simples. Cada tipo de dado espacial, assim como dados comuns podem ser utilizados em operações como exemplo, medir distância, perímetro, área e etc.

Modelos de Dados

Os dados espaciais podem ser classiAcados em dois grandes modelos formais de representações para trabalharmos em nível conceitual da modelagem de dados, geo-objetos e geo-campos.

- ➔ **Geo-objetos:** No modelo geo-objetos são representadas coleções de entidades distintas ou semelhantes com delimitações bem especificadas, ou seja, são individualizáveis no mundo real e cada componente tem sua identidade e seu atributo. Divisões políticas e terrenos em um loteamento são alguns exemplos de geo-objetos. O uso de coleções de geo-objetos é muito frequente em bancos de dados espaciais, pois é muito proveitoso tratar geo-objetos parecidos de forma consistente.
- ➔ **Geo-campo:** Neste modelo o espaço é contínuo sem delimitações, onde dentro do geo-campo é possível existir inúmeros geo-objetos ou mesmo todo o geo-campo estar dentro de um único geo-objeto. São fenômenos de variação contínua no espaço e tempo. Tipo de vegetação, solo e relevo, são exemplos de geo-campo. Os geo-campos podem ser divididos em partes e ainda assim manter sua propriedade essencial.

PosgreSQL

No postgreSQL foi realizada a implementação do PostGIS que é a extensão que manipula dados espaciais. O PostGIS (Ramsey, 2002) estende o PostgreSQL, seguindo as especificações da SFSSQL. O PostgreSQL é um sistema de gerência de banco de dados objeto-relacional, gratuito e de código fonte aberto (Stonebraker et al, 1990). Foi desenvolvido a partir do projeto Postgres, iniciado em 1986, na Universidade da Califórnia em Berkeley.

O PostgreSQL é um sistema gerenciador de banco de dados objeto-relacional, gratuito e de código fonte aberto, desenvolvido a partir do projeto Postgres, iniciado em 1986, na Universidade da Califórnia em Berkeley, sob a liderança do professor Michael Stonebraker. Em 1995, quando o suporte a SQL foi incorporado, o código fonte foi disponibilizado na Web (http://www.postgresql.org). Desde então, um grupo de desenvolvedores vem mantendo e aperfeiçoando o código fonte sob o nome de PostgreSQL.

Em sua versão de distribuição oficial, ele apresenta tipos de dados geométricos (Figura 8.9), operadores espaciais simples e indexação espacial através de uma R-Tree nativa ou através de R-Tree implementada no topo do mecanismo de indexação GiST (Hellerstein et al, 1995).

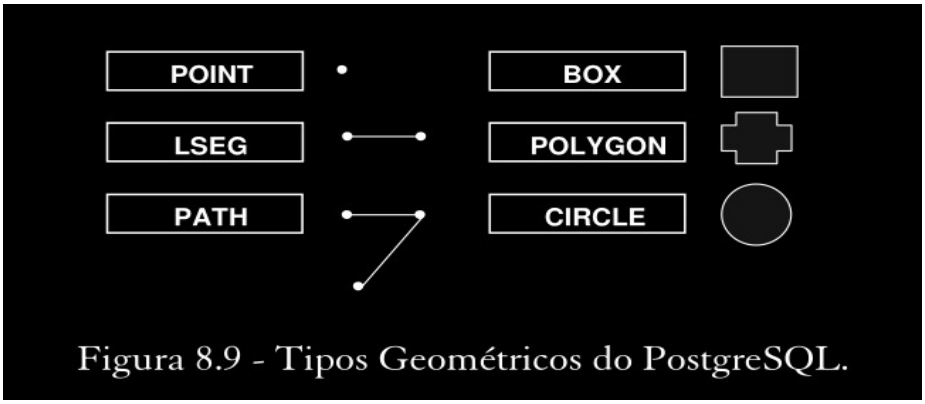


Figura 8.9 - Tipos Geométricos do PostgreSQL.

A implementação da R-Tree nativa possui uma severa limitação em seu uso, uma coluna do tipo polígono não pode exceder 8Kbytes. Na prática, é muito comum um SIG manipular dados representados, por exemplo, por polígonos maiores do que 8Kbytes cada, o que torna o uso desse índice inviável. Uma alternativa é o uso da segunda R-Tree.

O método de indexação chamado GiST foi introduzido por Hellerstein et al (1995) e implementado no PostgreSQL. Atualmente, ele é mantido por Signaev e 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 pode fornecer tanto as funcionalidades de uma B-Tree quanto de uma R-Tree e suas variantes. A principal vantagem desse índice é a possibilidade de definição do seu “comportamento”.

Quanto aos poucos operadores espaciais existentes, estes realizam a computação apenas sobre o retângulo envolvente das geometrias e não diretamente nelas. Já os tipos de dados simples, como polígonos, não permitem a representação de buracos, não existindo também geometrias que permitam representar objetos mais complexos como os formados por conjuntos de polígonos.

Portanto, a integração de um SIG através desses tipos geométricos requer muito esforço – implementação de novas operações espaciais como união, interseção, testes topológicos sobre a geometria exata, definição de um modelo de suporte a polígonos com buracos, entre outras.

Mas, como dito anteriormente, um dos pontos fortes desse SGBD é seu potencial de extensibilidade, o que possibilitou o desenvolvimento de uma extensão geográfica mais completa, chamada PostGIS. Antes de entrar em maiores detalhes dessa extensão, apresentaremos um pouco do mecanismo de extensibilidade para que o leitor possa entender melhor o núcleo da extensão PostGIS.

Extensibilidade do PostgreSQL

O mecanismo de extensibilidade do PostgreSQL permite incorporar capacidades adicionais ao sistema de forma a torná-lo mais flexível para o gerenciamento de dados para cada classe de aplicação. No caso dos SIG, isso significa a possibilidade do desenvolvimento de uma extensão geográfica capaz de armazenar, recuperar e analisar dados espaciais. A seguir serão apresentados alguns passos envolvidos na criação de uma extensão do PostgreSQL. Os exemplos foram adaptados do tipo de dados POLYGON existente na distribuição oficial.

A comunidade de software livre vêm desenvolvendo a extensão espacial PostGIS, construída sobre o PostgreSQL. Atualmente, a empresa Refractions Research Inc (http://postgis.refractions.net) mantém a equipe de desenvolvimento dessa extensão, que segue as especificações da SFSSQL.

Tipos de Dados Espaciais

A Figura 8.10 ilustra os tipos espaciais suportados pelo PostGIS e embutidos na SQL do PostgreSQL.

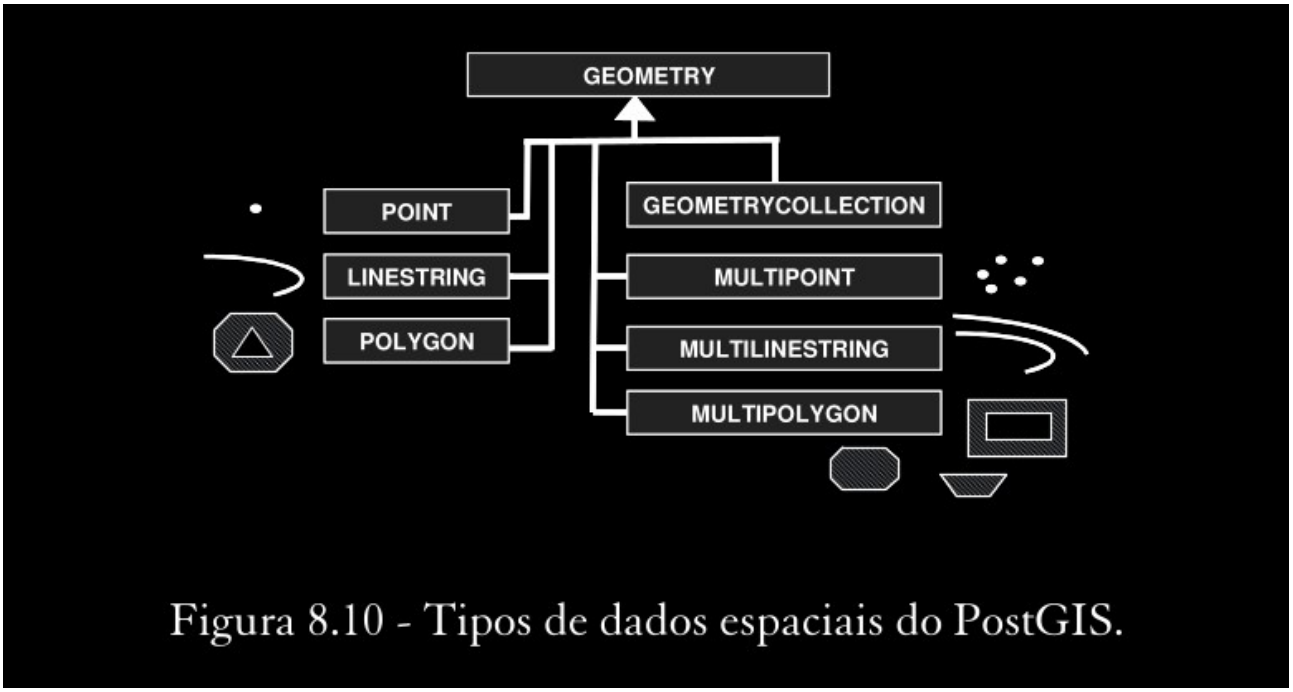


Figura 8.10 - Tipos de dados espaciais do PostGIS.

Esses tipos possuem a seguinte representação textual:

- Point: (0 0 0)
- LineString: (0 0, 1 1, 2 2)
- Polygon: ((0 0 0, 4 0 0, 4 4 0, 0 4 0, 0 0 0), (1 0 0,...), …)
- MultiPoint: (0 0 0, 4 4 0)
- MultiLineString: ((0 0 0, 1 1 0, 2 2 0), (4 4 0, 5 5 0,6 6 0))
- MultiPolygon: (((0 0 0, 4 0 0, 4 4 0, 0 4 0, 0 0 0),(...), ...), …)
- GeometryCollection: (POINT(2 2 0), LINESTRING((4 4 0, 9 9 0))

Como exemplo, mostraremos os comandos em SQL para gerar tabelas para armazenar os atributos e as geometrias3:

- Dos distritos de São Paulo, mostrados na Figura 8.1.

```
CREATE TABLE distritossp
( cod               SERIAL,
sigla               VARCHAR(10),
denominacao         VARCHAR(50),
PRIMARY KEY         (cod)
);
SELECT AddGeometryColumn('terralibdb', 'distritossp',
'spatial_data', -1, 'POLYGON', 2);
```

- Dos bairros de São Paulo, mostrados na Figura 8.2.

```
CREATE TABLE bairrossp
( cod                SERIAL,
bairro               VARCHAR(40),
distr                VARCHAR(40),
PRIMARY KEY          (cod)
);
SELECT AddGeometryColumn('terralibdb',
'bairrossp', 'spatial_data', -1,'POINT', 2);
```

Figura 8.1 – Mapa de Distritos da Cidade de São Paulo.

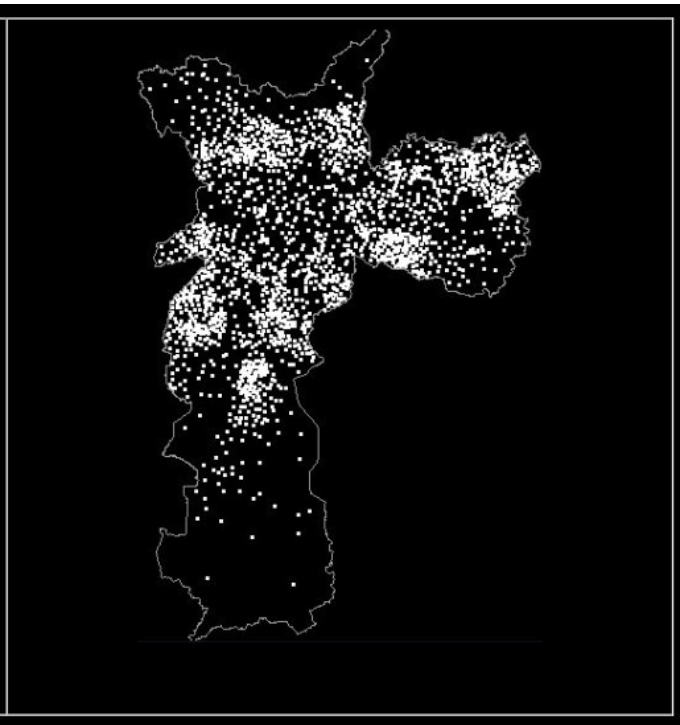


Figura 8.2 – Mapa de Bairros da Cidade de São Paulo.

- Do mapa de drenagem, mostrado na Figura 8.3:

```
CREATE TABLE drenagemsp
(
    cod                  SERIAL,
    classe               VARCHAR(255) NULL,
    PRIMARY KEY          (cod)
);
SELECT AddGeometryColumn('terralibdb', 'drenagemsp',
'spatial_data', -1, 'LINESTRING', 2);
```

- Do mapa de municípios da grande São Paulo (Figura 8.4)

```
CREATE TABLE grande_sp
( cod                SERIAL,
nomemunicp           VARCHAR(255) NULL,
população            INTEGER,
PRIMARY KEY          (cod)
);
SELECT AddGeometryColumn('terralibdb', 'grande_sp',
'spatial_data', -1, 'POLYGON', 2);
```

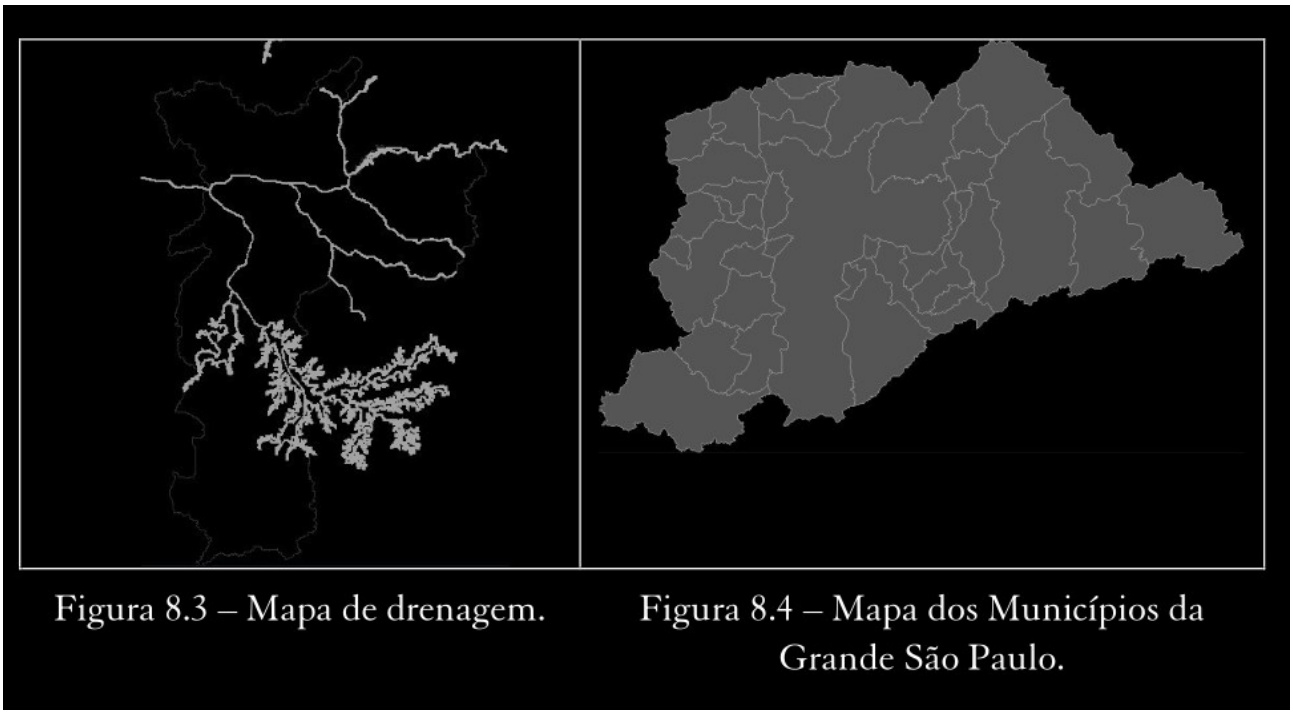


Figura 8.3 – Mapa de drenagem.

Figura 8.4 – Mapa dos Municípios da Grande São Paulo.

Observe que a criação de uma tabela com tipo espacial é construída em duas etapas. Na primeira, definimos os atributos básicos (alfanuméricos) e na segunda, usamos a função AddGeometryColumn para adicionar a coluna com o tipo espacial. Essa função implementada no PostGIS e especificada no OpenGIS, realiza todo o trabalho de preenchimento da tabela de metadado “geometry_columns”. Os parâmetros dessa função são:

- nome do banco de dados;
- nome da tabela que irá conter a coluna espacial;
- nome da coluna espacial;
- sistema de coordenadas em que se encontram as geometrias da
- tabela;
- tipo da coluna espacial, que serve para criar uma restrição que
- verifica o tipo do objeto sendo inserido na tabela;
- dimensão em que se encontram as coordenadas dos dados.

As tabelas de metadado do PostGIS seguem as especificações da SFSSQL e são representadas pelas seguintes tabelas:

Tabela 8.1 – Tabela de metadado do sistema de coordenadas

spatial_ref_sys		
Attribute	Type	Modifier
srid	INTEGER	PK
auth_name	VARCHAR(256)	
auth_srid	INTEGER	
srtext	VARCHAR(2048)	
proj4text	VARCHAR(2048)	

Tabela 8.2 – Tabela de metadado das tabelas com colunas espaciais

geometry_columns		
Attribute	Type	**Modifier**
f_table_catalog	VARCHAR(256)	PK
f_table_schema	VARCHAR(256)	PK
f_table_name	VARCHAR(256)	PK
f_geometry_column	VARCHAR(256)	PK
coord_dimension	INTEGER	
srid	INTEGER	FK
type	VARCHAR(30)	

Após criar as tabelas de dados, podemos inserir as informações usando o comando SQL INSERT. Para isso, podemos usar a representação textual das geometrias em conjunto com a função GeometryFromText que recebe a representação textual e mais o sistema de coordenadas em que se encontra a geometria:

INSERT INTO bairrossp (bairro, spatial_data) VALUES ('JARDIM DOS EUCALIPTOS', GeometryFromText('POINT(321588.628426 7351166.969244)', -1));

INSERT INTO drenagemsp (classe, spatial_data) VALUES ('RIOS', GeometryFromText ('LINESTRING(344467.895137 7401824.476217, 344481.584686 7401824.518728, 344492.194756 7401825.716359,…)', -1));

INSERT INTO distritossp (denominacao, sigla, spatial_data) VALUES('MARSILAC', 'MAR', GeometryFromText('POLYGON((335589.530575 7356020.721956, 335773.784959 7355873.470174, …))', -1));

Podemos também recuperar as informações em cada tabela. Por exemplo, o comando abaixo seleciona a linha do bairro "Vila Mariana":

SELECT bairro, AsText(spatial_data) geom FROM bairrossp WHERE
bairro = 'VILA MARIANA';

```
Resultado:
    bairro     |    geom
---------------+-----------------------------------
 VILA MARIANA | POINT(334667.138663 7388890.076491)
(1 row)       |
```

Aqui, empregamos a função AsText para obter a representação textual, pois a partir das versões mais recentes o PostGIS utiliza o formato binário do OpenGIS (WKB) como o padrão nas consultas.

As colunas com tipos espaciais podem ser indexadas através de uma R-Tree construída no topo do GiST. A sintaxe básica para criação de um índice é a seguinte:

CREATE INDEX sp_idx_name ON nome_tabela
USING GIST (coluna_geometrica GIST_GEOMETRY_OPS);

Para as tabelas do nosso exemplo, poderíamos construir os seguintes índices espaciais:

CREATE INDEX sp_idx_bairros ON bairrossp USING GIST
(SPATIAL_DATA GIST_GEOMETRY_OPS)

CREATE INDEX sp_idx_bairros ON distritossp USING GIST
(SPATIAL_DATA GIST_GEOMETRY_OPS)

CREATE INDEX sp_idx_bairros ON drenagemsp USING GIST
(SPATIAL_DATA GIST_GEOMETRY_OPS)

```
CREATE INDEX sp_idx_bairros ON grande_sp USING GIST
(SPATIAL_DATA GIST_GEOMETRY_OPS)
```

Os índices espaciais são usados em consultas que envolvam predicados espaciais, como no caso de consultas por janela, onde um retângulo envolvente é informado e as geometrias que interagem com ele devem ser recuperadas rapidamente.

O operador && pode ser usado para explorar o índice espacial. Por exemplo, para consultarmos os municípios da grande São Paulo que interagem com o retângulo envolvente de coordenadas: 438164.882699, 7435582.150681 e 275421.967006, 7337341.000355, podemos construir a seguinte consulta:

```
SELECT * FROM grande_sp
WHERE 'BOX3D(438164.882699 7435582.150681, 275421.967006
7337341.000355)' ::box3d && spatial_data);
```

Com o uso do operador &&, apenas alguns registros precisarão ser pesquisados para responder à pergunta acima.

Consultas Espaciais

Outro grande destaque desta extensão é o grande número de operadores espaciais disponíveis, entre alguns deles podemos citar:

- Operadores topológicos conforme a Matriz de 9-Interseções DE:
  equals(geometry, geometry)
  disjoint(geometry, geometry)
  intersects(geometry, geometry)
  touches(geometry, geometry)
  crosses(geometry, geometry)
  within(geometry, geometry)
  overlaps(geometry, geometry)
  contains(geometry, geometry)
  relate(geometry, geometry): retorna a matriz de intersecção.
- Operador de construção de mapas de distância:
  buffer(geometry, double, [integer])
- Operador para construção do Fecho Convexo:
  convexhull(geometry)
- Operadores de conjunto:
  intersection(geometry, geometry)
  geomUnion(geometry, geometry)
  symdifference(geometry, geometry)
  difference(geometry, geometry)
- Operadores Métricos:
  distance(geometry,geometry)
  area(geometry)
- Centróide de geometrias:
  Centroid(geometry)
- Validação (verifica se a geometria possui auto-interseções):
  isSimple(geometry)

O suporte aos operadores espaciais é fornecido através da integração do PostGIS com a biblioteca GEOS (Geometry Engine Open Source) (Refractions, 2003). Essa biblioteca é uma tradução da API Java JTS (Java Topology Suite) (Vivid Solutions, 2003) para a linguagem C++. A JTS é uma implementação de operadores espaciais que seguem as especificações da SFSSQL. Para exemplificar o uso desses operadores e funções, mostraremos os comandos em SQL para executar as consultas dos cenários de 1 a 5, apresentados no início desse capítulo.

Cenário 1 – Usando o operador touches, uma possível consulta seria:

```
SELECT d2.nomemunicp
FROM grande_sp d1, grande_sp d2
WHERE touches(d1.spatial_data, d2.spatial_data)
        AND (d2.nomemunicp <> 'SAO PAULO')
        AND (d1.nomemunicp = 'SAO PAULO');
```

Resultado:

nomemunicp	nomemunicp	nomemunicp
COTIA	JUQUITIBA	CAIEIRAS
ITAPECERICA DA SERRA	ITAQUAQUECETUBA	SAO BERNARDO DO CAMPO
EMBU-GUACU	EMBU	DIADEMA
SANTANA DE PARNAIBA	TABOAO DA SERRA	SAO CAETANO DO SUL
SANTO ANDRE	BARUERI	MAUA
GUARULHOS	CAJAMAR	FERRAZ DE VASCONCELOS
MAIRIPORA	OSASCO	POA

(21 rows)

Na consulta acima, o operador touches retorna verdadeiro caso as geometrias de d2 toquem na geometria de São Paulo. Esse é um exemplo de junção espacial entre duas relações (no nosso caso a mesma relação foi empregada duas vezes). Todas as geometrias da relação d1, com exceção da geometria São Paulo, foram avaliadas no teste topológico touches, pois o índice espacial não foi empregado. Em tabelas com grandes números de objetos, é importante a utilização desse índice. Ele pode ser explorado empregando-se o operador && em conjunto com os predicados da consulta anterior. Nossa consulta pode ser reescrita como:

```
SELECT d2.nomemunicp
FROM distritossp d1, distritossp d2
WHERE touches(d1.spatial_data, d2.spatial_data)
        AND (d2.nomemunicp <> 'SAO PAULO')
        AND (d1.spatial_data && d2.spatial_data)
        AND (d1.nomemunicp = 'SAO PAULO');
```

Cenário 2 – Novamente iremos empregar o operador touches:

```
SELECT grande_sp.nomemunicp
FROM distritossp, grande_sp
WHERE touches(distritossp.spatial_data, grande_sp.spatial_data)
        AND (distritossp.spatial_data && grande_sp.spatial_data)
        AND (distritossp.denominacao = 'ANHANGUERA');
```

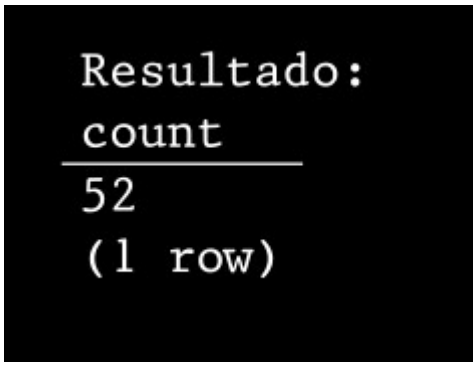
Resultado:

nomemunicp	nomemunicp
BARUERI	OSASCO
SANTANA DE PARNAIBA	CAIEIRAS
CAJAMAR	

(5 rows)

Cenário 3 – Para este cenário, podemos utilizar o operador contains:

```
SELECT COUNT(*)
FROM bairrossp pt, distritossp pol
WHERE contains(pol.spatial_data, pt.spatial_data)
        AND (pol.spatial_data && pt.spatial_data)
        AND pol.denominacao = 'GRAJAU';
```



Resultado:

count
52

(1 row)

Cenário 4 – Aqui empregaremos os operadores buffer e intersects:

```
SELECT grande_sp.nomemunicp
FROM grande_sp, drenagemsp
WHERE intersects(buffer(drenagemsp.spatial_data, 3000),grande_sp.spatial_data)
        AND drenagemsp.cod = 59;
```

Resultado:

Denominação	denominacao	denominacao
AGUA RASA	JAGUARA	SANTANA
ALTO DE PINHEIROS	JAGUARE	SAO DOMINGOS
BARRA FUNDA	LAPA	SE
BELEM	LIMAO	TATUAPE
BOM RETIRO	MOOCA	VILA FORMOSA
BRAS	PARI	VILA GUILHERME
CANGAIBA	PENHA	VILA LEOPOLDINA
CARRAO	PERDIZES	VILA MARIA
CASA VERDE	PIRITUBA	VILA MATILDE
CONSOLACAO	REPUBLICA	VILA MEDEIROS
FREGUESIA DO O	RIO PEQUENO	
JACANA	SANTA CECILIA	

(34 rows)

Cenário 5 – A resposta a essa pergunta poderia ser traduzida, de início, na seguinte consulta:

```
SELECT b1.bairro
FROM bairrossp b1, bairrossp b2
WHERE (distance(b1.spatial_data, b2.spatial_data) < 3000)
        AND b1.bairro <> 'BOACAVA'
        AND b2.bairro = 'BOACAVA' order by b1.bairro;
```

Resultado:

Bairro	bairro	bairro
ALTO DA LAPA	PINHEIROS	VILA INDIANA
ALTO DE PINHEIROS	SICILIANO	VILA IPOJUCA
BELA ALIANCA	SUMAREZINHO	VILA LEOPOLDINA
BUTANTA	VILA ANGLO BRASILEIRA	VILA MADALENA
JAGUARE	VILA HAMBURGUESA	VILA RIBEIRO DE BARROS
LAPA	VILA IDA	VILA ROMANA

(18 rows)

No entanto, podemos reescrever essa mesma consulta de forma mais eficiente, utilizando o índice espacial. A estratégia básica é montar um retângulo de 6Km por 6Km centrado no bairro BOACAVA, de forma que somente seja necessário calcular a distância para os pontos que estejam dentro do retângulo. Reescrevendo a consulta temos:

```
SELECT b1.nome, astext(b1.spatial_data)
FROM bairros b1, bairros b2
WHERE (distance(b1.spatial_data, b2.spatial_data) < 3000)
        AND (expand(b2.spatial_data, 3000) && b1.spatial_data)
        AND b1.nome <> 'BOACAVA'
        AND b2.nome = 'BOACAVA' ORDER BY b1.nome;
```

Fonte de Dados e consulta:
Instituto Nacional de Pesquisas Espaciais
INPE-12835-PRE/8125.

Portanto, pode-se concluir, que usar uma extensão de dados espaciais e geográficos, como a PostGIS, traz não só um ganho de performance como flexibilidade para lidar com dados espaciais e geográficos de uma forma geral. Permitindo, usar dados do usuário e implementações diversas aplicadas às necessidades e especificidades que se fizerem requerer.