Bancos de Dados Geográficos

Armazenamento

Extensões Espaciais ao SQL - PostGIS

Clodoveu Davis

DCC/UFMG

Escola de Verão DCC/UFMG

PostGIS

PostGIS

- Armazena objetos no formato WKT (Well Known Text), especificado pela OGC
  - Existem extensões para a terceira coordenada, chamadas genericamente de XYZ, XYM e XYZM
- A criação de tabelas não permite a inserção imediata de colunas de geometria
  - É necessário criar a tabela convencional primeiro, depois adicionar a coluna geométrica

PostGIS

- Exemplo

```
CREATE TABLE munic (
  ID NUMBER,
  NOME VARCHAR(20));
SELECT AddGeometryColumn('',
  'munic', 'geom', -1,
  'LINESTRING', 2);
```

  – Observar o formato anômalo desse select, sem cláusula FROM

PostGIS

- AddGeometryColumn(

```
<schema_name>, --opcional
<table_name>,  --nome tabela
<column_name>, --coluna geo
<srid>, --sist. ref. espacial
<type>, --tipo de geometria
<dimension> -- 2 ou 3
);
```

PostGIS

- Geometrias podem ser criadas a partir de constantes usando a função GeomFromText
- Isso permite a formulação de comandos INSERT

```
INSERT INTO munic (id, nome, geom)
VALUES (1001, 'Belo Horizonte',
GeomFromText('POLYGON(1 2, 2 3, 4 5, 6 7,
  1 2)', 29100);
```

PostGIS

- O string passado ao GeomFromText é exatamente a representação WKT
- A partir do WKT, foi especificada a representação WKB (Well Known Binary), mais compacta, que aparece nas colunas geométricas das tabelas PostGIS
- A representação WKT é muito simples, separando vértices com vírgulas e anéis em regiões com parênteses

PostGIS

- Exemplos WKT

```
POINT(123 456)
LINESTRING(0 0, 0 1, 1 1, 1 0)
POLYGON((0 0, 4 0, 4 4, 0 4, 0 0), (1 1, 2 1, 2 2, 1 1))
MULTIPOINT(0 0, 4 0, 4 4)
MULTILINESTRING((0 0, 1 1, 1 2, 2 2),(3 3, 4 4, 4 6, 6 6))
MULTIPOLYGON((0 0, 4 0, 4 4, 0 4, 0 0), (1 1, 2 1, 2 2, 1 1))
GEOMETRYCOLLECTION(POINT(123 456), LINESTRING(0 0, 0 1, 1 1, 1
  0), POINT(4 4))
```

- A representação não inclui o SRID, que vai ser informado no momento da inserção da geometria no banco

PostGIS

- Transformações entre WKT e WKB
  - bytea WKB = asBinary(geometry);
  - text WKT = asText(geometry);
  - geometry = GeomFromWKB(bytea WKB, SRID)
  - geometry = GeomFromText(text WKT, SRID)

PostGIS

- O PostGIS implementa extensões aos WKT/WKB, que são limitados na definição da OGC a geometrias 2D sem SRID
  - EWKB e EWKT
  - Formas “canônicas”: representações em hexadecimal
    - É o string que retorna após uma consulta à coluna geométrica se não for usado nenhum filtro ou função

PostGIS

- EWKT
  - `POINT(0 0 0)    -- ponto 3D`
  - `SRID=29100;POINT(123 300)    -- ponto 2D com SRID`
  - `POINTM(0 0 10)    -- XYM (2,5D)`
  - `POINT(0 0 0 0)      -- XYZM (3,5D?)`
  - Demais tipos incluindo vértices 3D

PostGIS

- EWKT e EWKB em comandos INSERT
  - `INSERT INTO pt_cotado (geom, ID) VALUES (GeomFromEWKT(‘SRID=29100;POINTM(-120.2 37.4 82.7)´), 1001);`

PostGIS

- Sistemas de referência espaciais
  - Parâmetros armazenados na tabela SPATIAL_REF_SYS

```
CREATE TABLE spatial_ref_sys (
srid INTEGER NOT NULL PRIMARY KEY,
auth_name VARCHAR(256),
auth_srid INTEGER,
srtext VARCHAR(2048),
proj4text VARCHAR(2048)
)
```

  - A coluna *srtext* é especificada em uma forma de WKT para SRS
  - A coluna *proj4text* é usada em uma biblioteca de projeção chamada Proj4

PostGIS

- Metadados
  - Existe uma tabela de metadados denominada geometry_columns para armazenar informação sobre as geometrias em uso

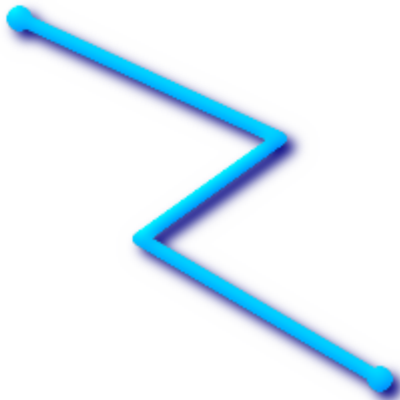
```
CREATE TABLE geometry_columns (
    f_table_catalog VARCHAR(256) NOT NULL,
    f_table_schema VARCHAR(256) NOT NULL,
    f_table_nam VARCHAR(256) NOT NULL,
    f_geometry_column VARCHAR(256) NOT NULL,
    coord_dimension INTEGER NOT NULL,
    srid INTEGER NOT NULL,
    type VARCHAR(30) NOT NULL
)
```

  - A coluna *type* informa o tipo de objeto para toda a coluna
    - Se for interessante restringir a coluna a apenas um tipo, usar o nome
    - Caso contrário, usar ‘GEOMETRY’

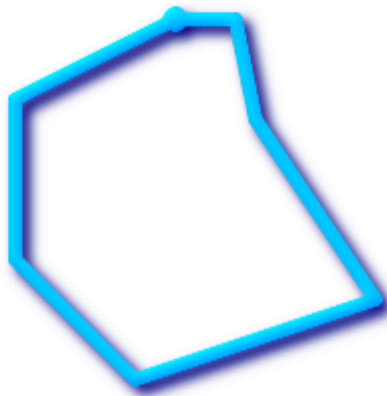
PostGIS

- Verificação de geometrias: o PostGIS, de acordo com o especificado pelo OGC, exige que as geometrias sejam *simples* e *válidas*
  - Simples: geometria que não tem pontos anômalos, como autointerseções, autotangências ou coincidências indesejadas
    - Aplicável a pontos e linhas
  - Válida: geometria (polígono) em que os anéis exteriores contém inteiramente os anéis interiores (a interseção, se houver, só pode ser em um ponto)

PostGIS


(a)

(b)


(c)

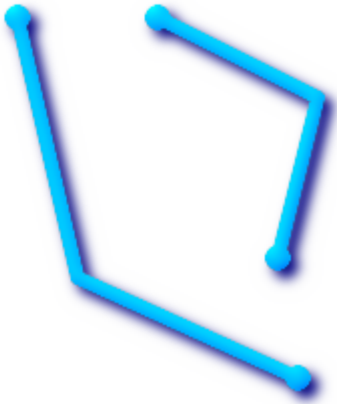
(d)

LINESTRINGS

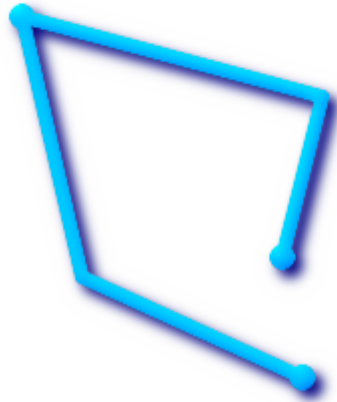
(a) e (c) são simples, (b) e (d) não são

(c) é chamado de anel linear (linear ring); linear rings têm que ter mais de 2 vértices (o último coincide com o primeiro, deve existir pelo menos mais um)

PostGIS



(a)



(b)

(c)

MULTILINESTRINGS

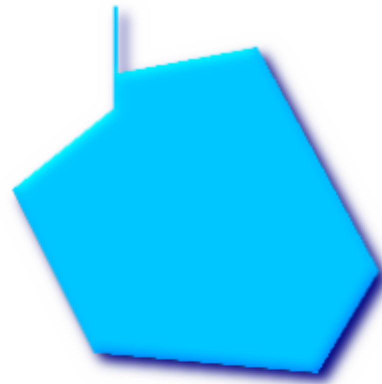
(a) e (b) são simples, (c) não é

(h)

(i)

(j)

(k)

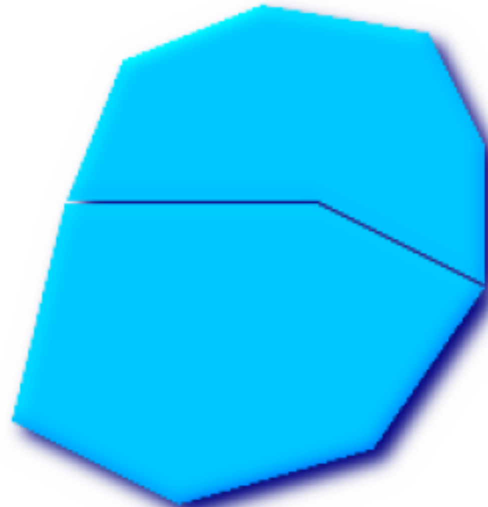


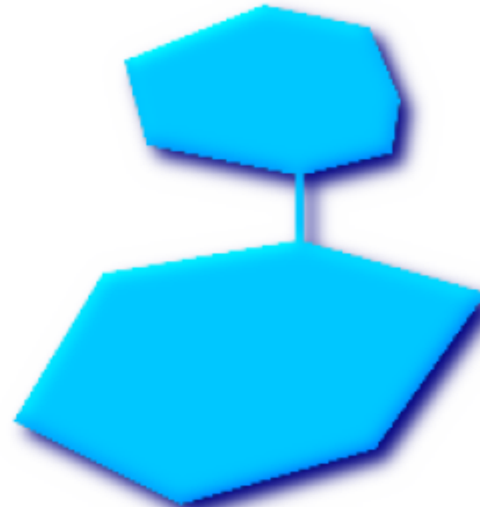
(l)

(m)

(h) and **(i)** are valid `POLYGON`s, **(j-m)** cannot be represented as single `POLYGON`s, but **(j)** and **(m)** could be represented as a valid `MULTIPOLYGON`.

PostGIS





(**n**) and (**o**) are not valid `MULTIPOLYGONs`.

MULTIPOLYGONS só são válidos se os interiores não tiverem interseção. As fronteiras podem ter interseção, mas apenas em pontos

PostGIS

- Verificação da validade de uma geometria
  - SELECT IsValid(geometria)
  - SELECT IsSimple(geometria)
- O PostGIS não aplica esse teste na entrada de cada geometria, pois isso pode demandar muito tempo de CPU
- Essa função não valida geometrias 3D (não OGC)
- Restrição de integridade para implementar isso:
  - ```
    ALTER TABLE tab ADD CONSTRAINT nome
    CHECK (IsValid(geom));
    ```
    ```

PostGIS

- Índices geográficos
  - O PostGreSQL suporta índices em B-Tree, R-Tree e GiST
    - B-Tree: índices convencionais
    - R-Tree: serviria para retângulos envolventes de objetos geográficos, mas a implementação do PostGreSQL não é considerada robusta o suficiente
    - GiST: Generalized Search Tree é uma forma de indexação genérica, que permite uma implementação melhorada da R-Tree
  - `CREATE INDEX nome ON tab USING GIST (geom);`
  - Após a criação de um índice espacial, é interessante forçar o PostGreSQL a coletar estatísticas para levar em conta esse índice na otimização das consultas geográficas
    - `VACUUM ANALYZE table column;`

PostGIS

- O PostGIS inclui um tipo chamado *Geography*, além do já discutido *Geometry*
  - *Geography* é baseado no esferóide, enquanto *Geometry* é baseado em um plano
    - Cálculos sobre o esferóide x cálculos sobre projeções
  - Poucas funções estão disponíveis para *Geography* por enquanto, e o único SRID admitido é o 4326 (WGS84 lon/lat)
  - A unidade de medida em *Geography* é o metro

PostGIS

- Geography: Não é necessário usar o AddGeometryColumn ou algo parecido

```
CREATE TABLE global_points (
  id SERIAL PRIMARY KEY,
  name VARCHAR(64),
  location GEOGRAPHY(POINT,4326)
);
```

PostGIS

- Indexação espacial: idem com Geography

```
CREATE INDEX global_points_gix ON global_points
  USING GIST ( location );
```

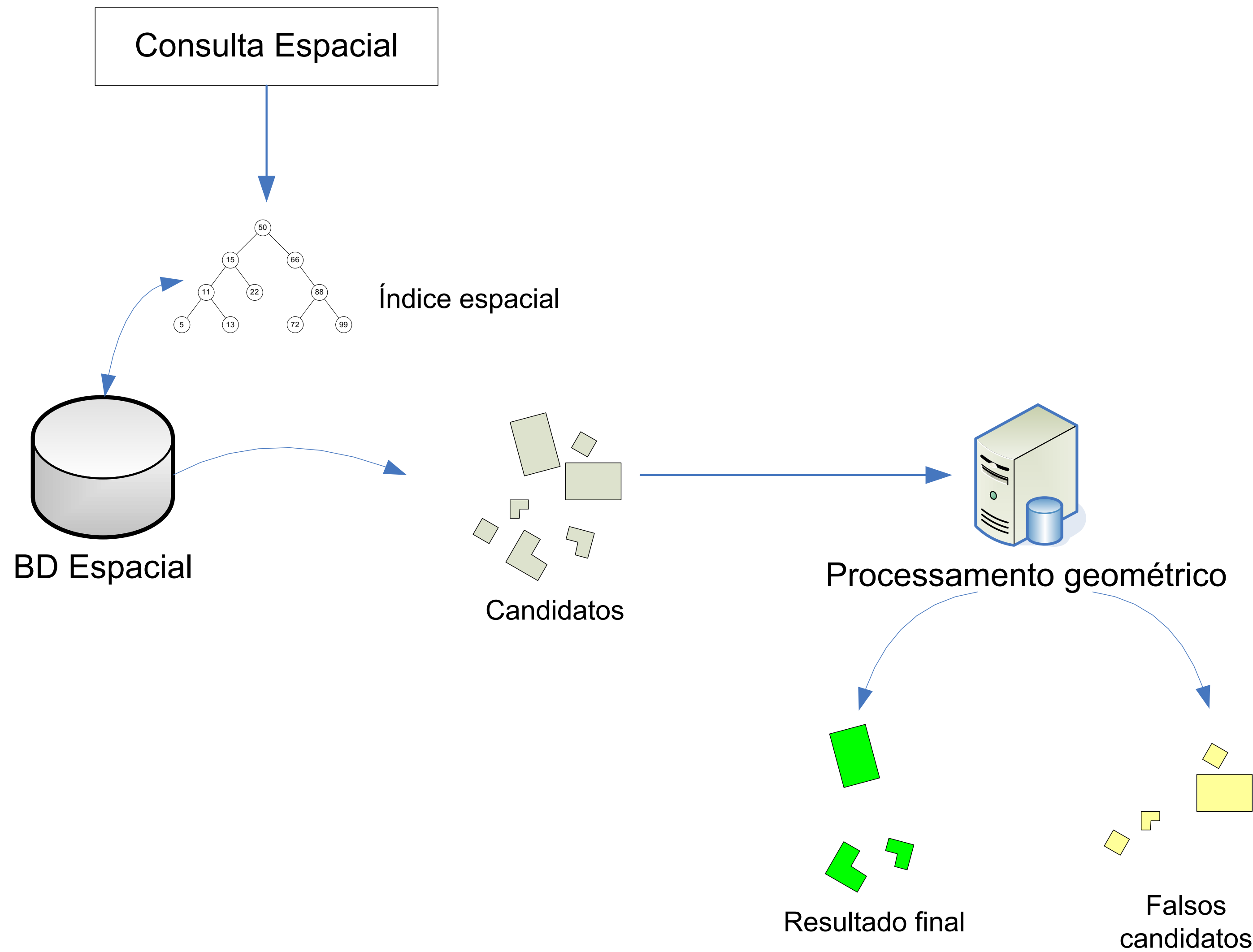
- Algumas funções de Geography:

```
ST_GeographyFromText('SRID=4326;POINT(-40,-20)');
ST_Area
ST_AsGML
ST_AsKML
ST_AsText
ST_Distance
ST_Covers
ST_CoveredBy
ST_Dwithin
ST_Intersects
ST_Length
```

PostGIS

- Além de Geometry e Geography, o PostGIS também oferece suporte a funções do padrão SQL/MM3 (Multimedia)

Processamento de consulta espacial



Consultas

- Extensões do SQL
- No PostGIS, as extensões incluem
  - Tabela spatial_ref_sys
  - Tabela geometry_columns
  - Funções geográficas para uso com SQL
    - DDL: funções de criação de tabelas e índices
    - DML: funções de processamento geográfico
    - Funções utilitárias, conversão de formatos, etc.

Consultas

SQL convencional: Recuperar a data de aniversário e o endereço do empregado cujo nome é 'João da Silva'

Consulta SELECT-PROJECT

```
SELECT  datanasc, endereco
FROM    empregado
WHERE   nome = 'João da Silva'
```

Consultas

- Nesse tipo de consulta, a cláusula WHERE contém um FILTRO
- O filtro pode ser espacial também, usando funções da extensão que retornem valor booleano ou compondo uma expressão

Consultas

- Exemplo (supondo que empregado seja representado usando pontos)

```
SELECT   datanasc, endereco
FROM     empregado
WHERE    ST_DISTANCE(geom,
         GeomFromText('POINT (697700
         7781000)', 29193))
         < 1000
```

- No exemplo, a função GeomFromText foi usada para produzir uma constante do tipo Geometry
- Para que ST_DISTANCE funcione, é necessário que ambas as geometrias estejam no mesmo sistema de projeção e coordenadas (no caso, 29193 = UTM fuso 23S)
- Se as geometrias estiverem em um sistema lat-long, a resposta será em GRAUS, o que é inutil, portanto é necessário converter para um sistema com coordenadas métricas

Consultas

- Exemplo 2 (supondo que empregado seja representado usando pontos em lat-long)

```
SELECT   datanasc, endereco
FROM     empregado
WHERE    ST_DISTANCE(
         ST_TRANSFORM(geom, 29193),
         GeomFromText('POINT (697700
         7781000)', 29193))
         < 1000
```

Consultas

- Exemplo 3 (supondo que o ponto de referencia seja dado em WGS84)

```
SELECT     datanasc, endereco
FROM       empregado
WHERE      ST_DISTANCE(
           ST_TRANSFORM(geom, 29193),
           ST_TRANSFORM(
           GeomFromText(‘POINT (-42.9981
           -19.2290)’, 4326)
           , 29193)
           < 1000
```

Consultas

Recuperar o nome e o endereço de todos os empregados que trabalham no departamento cujo nome é ‘Pesquisa’.

Consulta SELECT-PROJECT-JOIN

```
SELECT   nome, endereco
FROM     empregado e, departamento d
WHERE    e.numdep = d.numdep
AND      nomedep = ‘Pesquisa’
```

Consultas

- No exemplo, existe uma cláusula de filtro e uma cláusula de JUNÇÃO
- É necessário ter uma junção para cada par de tabelas; com n tabelas no FROM, tem-se n-1 cláusulas de junção
- Cláusulas de junção podem ser geográficas, usando funções apropriadas

Consultas

- Exemplo (município = polígono, aeroporto = ponto)

```
SELECT   nome_muni, pop_muni
FROM     municipio m, aeroporto a
WHERE    pop_muni < 100000
AND      ST_CONTAINS(m.geom, a.geom)
```

Consultas

- Exemplo 2

```
SELECT   m2.nome_muni, m2.pop_muni
FROM     municipio m1, municipio m2
WHERE    m1.nome_muni = 'Uberaba'
AND      ST_TOUCHES(m1.geom, m2.geom)
```

PostGIS

- Funções geométricas
  - ST_Distance(geom, geom)
  - ST_DWithin(geom, geom, distance)
  - ST_Centroid(geom)
  - ST_Area(geom)
  - ST_Length(geom)
  - ST_PointOnSurface(geom)
  - ST_Boundary(geom)
  - ST_Buffer(geom, distance, numCircleSegments)
  - St_ConvexHull(geom)

PostGIS

- Funções geométricas (cont)
  - ST_Intersection(geom, geom)
  - ST_Difference(geom, geom)
  - ST_Union(geom, geom)
    - Variação: ST_Union(GeomSet)
    - Variação: ST_MemUnion(GeomSet)
  - ST_SymDifference(geom, geom)

PostGIS

- ST_AsText(geom)
- ST_AsBinary(geom)
- ST_SRID(geom)
- ST_Dimension(geom)
- ST_Envelope(geom)
- ST_IsEmpty(geom)
- ST_IsSimple(geom)
  - True se não houverem autointerseções
- ST_IsClosed(geom)
- ST_IsRing(geom)
  - True se a curva for simples e fechada
- ST_NumGeometries(geom)
  - Número de elementos em uma coleção

PostGIS

- ST_GeometryN(geom, int)
  - Retorna a n-ésima geometria em uma coleção
- ST_NumPoints(geom)
  - Retorna o número de vértices do primeiro linestring na geometria
- ST_PointN(geom, int)
  - Retorna o n-ésimo vértice da geometria
- ST_ExteriorRing(geom)
- ST_NumInteriorRings(geom)
- ST_InteriorRing(geom, int)
- ST_EndPoint(geom)
- ST_StartPoint(geom)
- GeometryType(geom) / ST_GeometryType(geom)
  - Retorna string com o nome do tipo da geometria
- ST_X(geom)
- ST_Y(geom)
- ST_Z(geom)

PostGIS

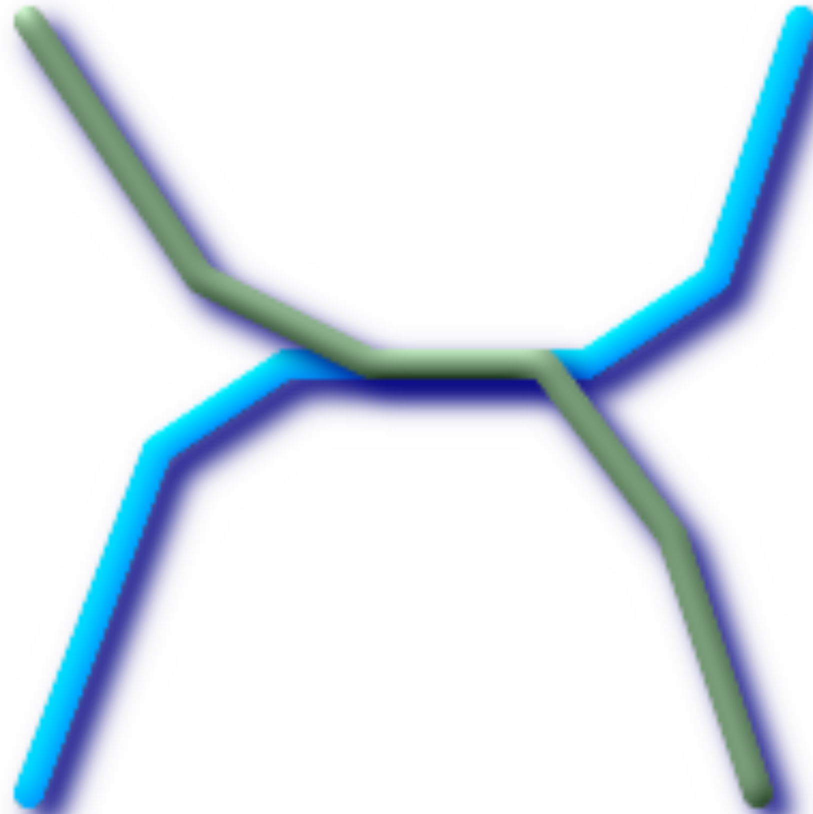
- Funções topológicas
  - Nomes ligeiramente diferentes da teoria de Egenhofer, mas aderentes aos padrões OpenGIS e SQL-MM
  - Usam a matriz de 9 interseções (ordem: interior, fronteira, exterior), numa versao estendida, chamada de **DE-9IM** (*Dimensionally extended 9-intersection matrix*)
  - As posições da matriz são preenchidas com
    - 0 -> interseção em ponto (0D)
    - 1 -> interseção em linha (1D)
    - 2 -> interseção em polígono (2D)
    - T -> interseção em ponto, linha ou polígono
    - F -> sem interseção
    - * -> indiferente

- Exemplo
- Matriz = 212101212 (leitura esq->dir, top->bottom)

	Interior	Boundary	Exterior
Interior	dim(...) = 2	dim(...) = 1	dim(...) = 2
Boundary	dim(...) = 1	dim(...) = 0	dim(...) = 1
Exterior	dim(...) = 2	dim(...) = 1	dim(...) = 2

PostGIS

- Exemplo
- Matriz = 1*1***1*



PostGIS

- Funções topológicas
  - ST_Equals(geom, geom)
  - ST_Disjoint(geom, geom)
  - ST_Intersects(geom, geom)
    - _ST_Intersects(geom, geom): idem, mas evita o uso do índice
    - == NOT disjoint(geom, geom) (ANYINTERSECT do Oracle)
  - ST_Touches(geom, geom)
  - ST_Crosses(geom, geom)
  - ST_Within(geom, geom)
  - ST_Overlaps(geom, geom)
  - ST_Contains(geom, geom)
  - ST_Covers(geom, geom)
  - ST_CoveredBy(geom, geom)
  - ST_Relate(geom, geom, intPatternMatrix)
  - ST_Relate(geom, geom)
    - Retorna a DE-9IM (dimensionally extended 9-intersection matrix)

Funções topológicas

- ST_Equals(geom1, geom2)
  - Retorna TRUE se as geometrias forem "espacialmente equivalentes"
- ST_Disjoint(geom1, geom2)
  - Retorna TRUE se as geometrias forem disjuntas
- ST_Intersects(geom1, geom2)
  - Retorna TRUE se as geometrias tiverem qualquer ponto de interseção
  - Usar _ST_Intersects para evitar o uso do índice espacial
  - Intersects == NOT disjoint

Funções topológicas

- ST_Touches(geom1, geom2)
  - Retorna TRUE se as geometrias se tocarem (4IM): interseção entre fronteiras é não vazia, interseção dos interiores é vazia
- ST_Crosses(geom1, geom2)
  - Retorna TRUE se as geometrias se cruzam, ou seja, se elas tiverem alguns pontos do interior em comum, mas não todos
  - Para evitar o uso do índice, usar _ST_Crosses
  - DE-9IM: T*T****** para ponto/linha, ponto/polígono e linha/polígono
  - DE-9IM: T*****T** para linha/ponto, polígono/ponto e polígono/linha
  - DE-9IM: 0******** para linha/linha

$$a.Crosses(b) \Leftrightarrow (dim(I(a) \cap I(b)) < max(dim(I(a)), dim(I(b)))) \wedge (a \cap b \neq a) \wedge (a \cap b \neq b)$$

Funções topológicas

- ST_Within(geom1, geom2)
  - Retorna TRUE se a geom1 está totalmente dentro da geom2
  - Para evitar o uso do índice, usar _ST_Within
  - DE-9IM: T*F**F***
- ST_Overlaps(geom1, geom2)
  - Retorna TRUE se as geometrias compartilharem algum espaço, forem da mesma dimensão, e uma não contiver inteiramente a outra
  - Não é o mesmo que ST_Intersects

Funções topológicas

- ST_Contains(geom1, geom2)
  - Retorna TRUE se a geom1 contiver espacialmente a geom2, ou seja, nenhum ponto de geom2 pode estar no exterior de geom1, e pelo menos um ponto de geom2 está no interior de geom1
  - Equivale a ST_Within(geom2, geom1)
- ST_ContainsProperly(geom1, geom2)
  - Retorna TRUE se geom2 interceptar o interior de geom1, mas não sua fronteira ou o exterior
  - DE-9IM: T**FF*FF*
  - Este é o relacionamento CONTAINS do 4IM

Funções topológicas

- ST_Covers(geom1, geom2)
  - Retorna TRUE se nenhum ponto de geom2 estiver no exterior de geom1
  - É uma função definida em 4IM, mas curiosamente não é parte do padrão OGC; apesar disso, existe no Oracle e no PostGIS
- ST_CoveredBy(geom1, geom2)
  - Inverso de ST_Covers

Funções topológicas

- ST_Relate(geom1, geom2)
  - Retorna a DE-9IM que ocorre entre as geometrias
- ST_Relate(geom1, geom2, matriz)
  - Retorna TRUE se o relacionamento existente entre as duas geometrias atender ao especificado na matriz

Funções topológicas

- ST_OrderingEquals(geom1, geom2)
  - Retorna TRUE se as geometrias forem iguais e seus vértices estiverem dispostos na mesma ordem/direção
  - ST_Reverse(geom) retorna uma geometria com a sequência de vértices invertida
- ST_DWithin(geom1, geom2, distância)
  - Retorna TRUE se as geometrias estiverem à distância especificada uma da outra
- ST_DFullyWithin(geom1, geom2, distância)
  - Retorna TRUE se as geometrias estiverem inteiramente dentro da distância especificada uma da outra

PostGIS

- Operadores
  - A = B -> true se os MBRs coincidirem
  - A &< B -> true se o MBR de A intercepta ou está à esquerda do de B
  - A &> B -> true se ocorrer o inverso
  - A << B -> true se o MBR de A estiver à esquerda do de B
  - A >> B -> true se ocorrer o inverso
  - Outros operadores para MBR: abaixo (&<|, <<|), acima (|&>, |>>), iguais (~=), contém (~), contido (@), overlaps (&&)

PostGIS

- ST_Distance_Sphere(point, point)
  - Distância geodésica aproximada, considerando o planeta como se fosse esférico
- ST_Distance_Spheroid(point, point, spheroid)
  - Distância linear entre dois pontos lat/lon para um esferóide em particular, dado em um string:
    - ‘SPHEROID[“<NAME>”,<SEMI-MAJOR AXIS>,<INVERSE FLATTENING>]’
- Uma alternativa é usar reprojeção (conversão para outro sistema de coordenadas, e cálculo de distância euclidiana)
  - ST_Transform(geom, novoSRID)

PostGIS

- ST_AsGML([version,] geom [, precision])
  - Version = 2 ou 3
  - Precision: default é 15 digitos significativos
- ST_AsKML([version,] geom [,precision])

PostGIS

- ST_MakePolygon(linestring [,linestring[]])
  - Cria um polígono a partir de várias linhas
- ST_BuildArea(geom)
  - Cria um polígono a partir de um objeto de linha
- ST_Polygonize(geomSet)
  - Cria uma GeometryCollection
- ST_Collect(geomSet) / ST_Collect(geom, geom)
  - Retorna uma GeometryCollection ou um objeto Multi

PostGIS

- ST_Summary(geom)
- ST_ndims(geom)
- ST_npoints(geom)
- ST_nrings(geom)
- ST_isvalid(geom)
- ST_box2d(geom)
- ST_box3d(geom)