# 《交通大数据技术》

**马晓磊**

**交通科学与工程学院**

**2024年**

# 空间数据库

# 空间数据的扩展SQL

动机

- SQL具有简单的原子数据类型，如整数、日期和字符串；
- 不方便空间数据和查询
  - 空间数据（例如：多边形）是复杂的
  - 空间操作：拓扑、欧几里得、方向性、度量

SQL 3允许用户定义数据类型和操作

- 空间数据类型和操作可以添加到SQL3

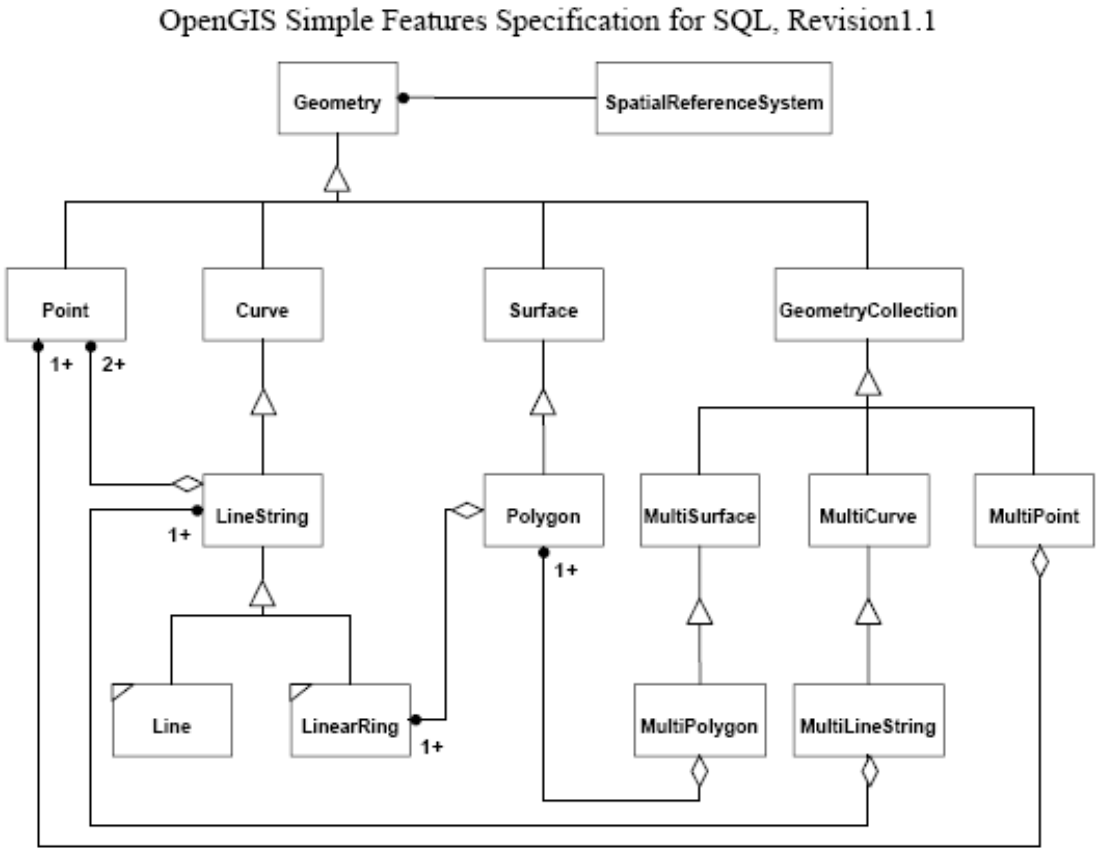
开放地理空间联盟 (Open Geospatial Consortium， OGC)标准

- 六种空间数据类型
- 几种空间操作
- 由主要供应商支持，如ESRI、Intergraph、Oracle、IBM...

# OGIS空间数据模型

由基类几何和四个子类组成：

- 点、曲线、曲面和几何集合



# 矢量数据类型

常用的矢量数据格式主要包括以下六类：

- 点(Point)
- 多点(MultiPoint)
- 线(LineString)
- 多线(MultiLineString)
- 面(Polygon)
- 多面(MultiPolygon)

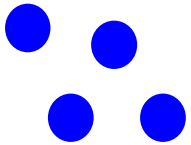
# 矢量数据类型

## 点（Point)

单独一个坐标点构成的空间实体。

## 多点(MultiPoint)

由多个坐标点构成的一个空间实体。



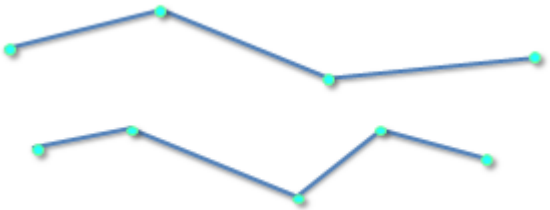
# 矢量数据类型

## 线(LineString)

由一条线构成的空间实体。

## 多线(MultiLineString)

有多条线构成的一个空间实体。



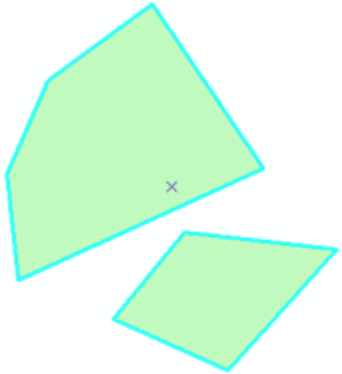
# 矢量数据类型

## 面(Polygon)

Polygon具有1个外环和0个或n个内环构成。

## 多面(MultiPolygon)

由多个相离的Polygon构成，并没有一个外环。



# OGIS空间数据模型

**操作分为三类：**

- 应用于所有几何类型
  - 空间参考、包络、导出、IsSimple、Boundary
- 拓扑关系的谓词
  - 相等、不相交、相交、相接、交叉、包含于、包含
- 空间数据分析
  - 距离、缓冲区、并集、交集、差集、凸包

# 利用SQL/OGIS实现空间操作

## 基本函数

SpatialReference（）：返回基础坐标系。

Envolope（）：返回最小边界矩形（MBR）。

Export（）：以不同的表示形式返回几何体

IsEmpty（）：如果几何体为空集，则返回true。

IsSimple（）：如果没有自相交，则返回true

Boundary（）：返回几何体的边界。

# 利用SQL/OGIS实现空间操作

## 拓扑和集合比较运算

Equal： 判断两个几何对象是否相同

Disjoint：判断两个几何对象是否分离

Intersection：判断两个几何对象是否相交

Touch：判断两个几何对象的边缘是否接触

Cross：判断两个几何对象是否互相穿过

Within：判断A是否被B包含

Contains：判断A是否包含B

Overlap：判断两个几何对象是否是重叠

# 利用SQL/OGIS实现空间操作

## 空间分析

Distance：返回最短距离

Buffer：缓冲区

ConvexHull：获取多几何对象的最小外接对象

Intersection：获取两个几何对象相交的部分

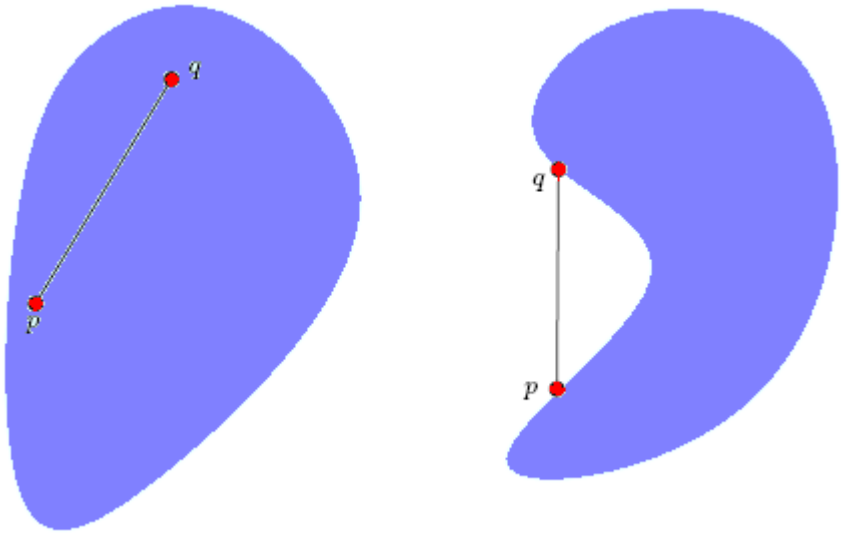
Union：获取两个几何对象全集

Difference：从A去除和B相交的部分后返回

SymmDif：获取两个几何对象不相交的部分

# Convex Hull（凸包）

一组点的凸包是包含这些点的所有凸集的交集。点集是凸的当且仅当对于S中的每对点p，q，线段pq完全包含在S中。




左边是凸集，右边是非凸集

# 为什么要为 GIS 使用数据库?

严格地说，GIS不是数据库系统。GIS可以被连接到DBMS。

GIS不能有效地管理大量的非空间数据（例如，在政府部门级别上）。

它们缺乏特殊的查询能力（它们提供了一种限定形式的预定义查询）。

它们缺乏用于快速外部数据访问的索引结构（它们使用内存技术）。

# 为GIS选择数据库?

数据库管理系统的选择：

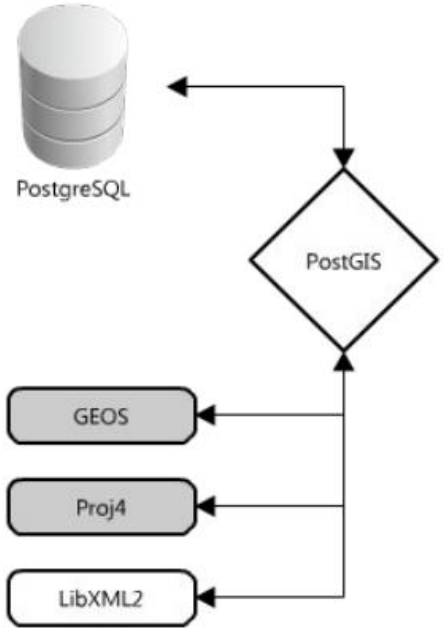
- 商业（Oracle，DB2）或
- 开源（PostgreSQL，MySQL）。

我们将使用PostGIS空间扩展的PostgreSQL

PostgreSQL是一个对象关系数据库系统（ORDBMS）。

主要用C编写

使用一些外部库



# 其他空间DBMS

Oracle Spatial

ESRI ArcSDE

IBM DB2

Microsoft SQLServer（>2008）

SpatiaLite

Ingres




# PostGIS能做什么?

许多 PostGIS 功能通过SQL提供

符合OGC1简单特性规范

- 坐标变换
- 识别（SRID）
- 缓冲区
- 相接
- 交叉
- 重叠
- 包含于
- 包含
- 面积
- 长度
- 表面上的点
- 返回几何图形为SVG

# PostGIS能做什么?

PostGIS支持符合OGC标准的简单特性的几何图形类型。

- 点（50 100）
- 线串（10 10，20 20）
- 多边形（（0 0，5 5，5 0，0 0））
- 多点（（1 1），（0 0））
- 多行线（...）
- 多多边形（...）

# PostgreSQL

PostgreSQL 本身提供了RDBMS的主要特性。包括其他高级功能，如：

- 继承
- 函数
- 约束
- 触发器
- 规则
- 事务完整性

允许“OOlike”编程风格

# PostgreSQL/PostGIS

数据以相对简单的格式存储，属性和几何图形存储在一个表中。它可以作为已知文本（WKT）查看，也可以使用asBinary（the_geom）函数以图形方式显示。


| name | city | hrs | status | st_fed | the_geom |
|---|---|---|---|---|---|
| Brio Refining | Friendswood | 50.38 | active | Fed | SRID=32140;POINT(968024.87474318 4198600.9516049) |
| Crystal Chemical | Houston | 60.9 | active | Fed | SRID=32140;POINT(932279.183664999 4213955.37498466) |
| North Cavalcade | Houston | 37.08 | active | Fed | SRID=32140;POINT(952855.717021537 4223859.84524946) |
| Dixie Oil Processors | Friendswood | 34.21 | active | Fed | SRID=32140;POINT(967568.655313907 4198112.19404211) |
| Federated Metals | Houston | 21.28 | active | State | SRID=32140;POINT(961131.619598681 4220206.32109146) |

# 它是如何工作的?

空间数据是使用特定投影的坐标系存储的。

该投影使用空间引用标识号（SRID）进行引用

该标号与另一个包含所有可用空间参考系的表（Spatial_ref_sys）相关。

这允许数据库知道每个表所处的投影，如果需要，可以从这些表中重新投影以进行计算或与其他表联接。

# 坐标系基础

WGS84坐标系：即地球坐标系，国际上通用的坐标系。

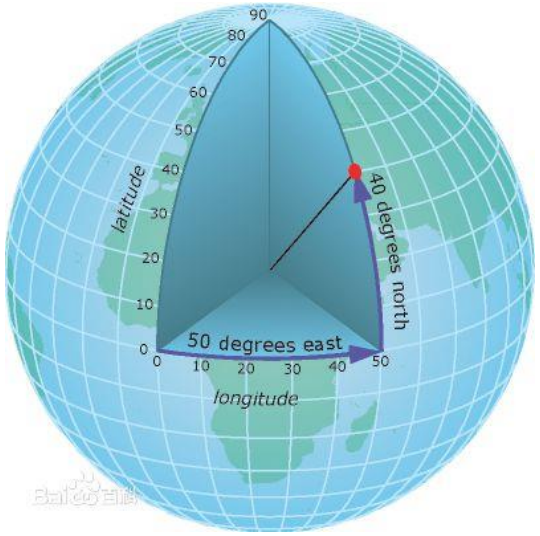
GCJ02坐标系：即火星坐标系，WGS84坐标系经加密后的坐标系。

高德地图

BD09坐标系：即百度坐标系，GCJ02坐标系经加密后的坐标系。

百度独有

Web墨卡托坐标系




# 为什么要进行投影

地球是三维的

地图（屏幕）是二维的

地理坐标系统是在三维（基准面）上的定位方法

投影使得三维转成二维

三维转成二维会产生变形

## 空间引用标识符 (SRID)

每个空间实例都有一个空间引用标识符 (SRID)。SRID 对应于基于特定椭圆体的空间参照系统，可用于平面球体映射或圆球映射。

## 常用的坐标系统对应的SRID

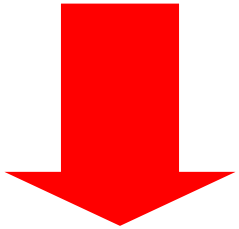
WGS84坐标系（srid=4326）

Web墨卡托坐标系（srid=3857）

# 坐标投影

SRID=3005; MULTILINESTR((1004687.04355194594291.
053764096,1004729.74799931 594258.821943696))

1983年北美基准
SRID 3005=NAD83
SRID 4326=WGS84



SRID=4326; MULTILINESTR((125.934150.364070000001,
-125.9335 50.36378))

一个表的坐标可以转换成另一个表的坐标。这允许每个表中的“几何体”匹配。在PostGIS中相对容易实现

# 单选题 1分

设置

WKB编码在PostGIS中用于什么目的?

A 存储文本格式的SQL命令

B 以二进制形式存储几何对象

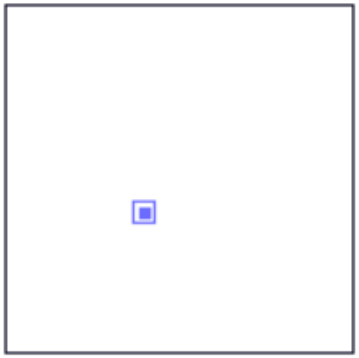
C 加密数据库连接

D 优化文本查询性能

提交

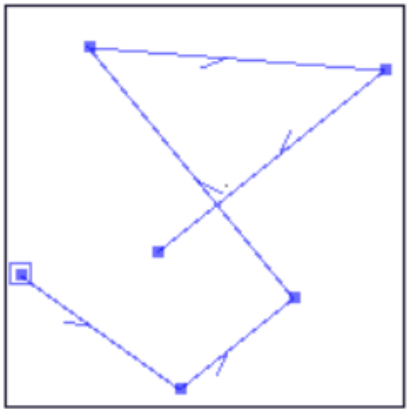
# 几何体：点Point

```
POINT (10 10)
```



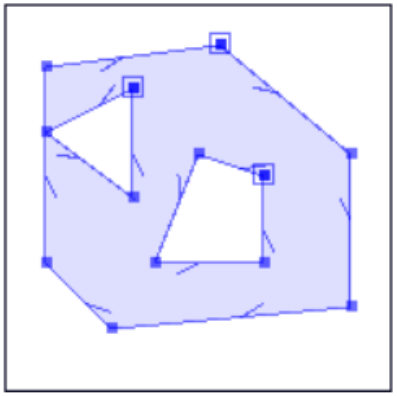
# 几何图形：线串LineString

```
LINESTRING
 (
  0 5, 5 1, 9 4, 2 14, 14 13, 4 4
 )
```



# 几何：多边形Polygon

```
POLYGON
(
 (9 13,13 9,13 3,4 2,1 4,1 12, 9 13),
 (5 11,5 6,1 9,5 11),
 (10 7, 10 4, 6 4, 8 8, 10 7)
)
```



1) 强制第一个环是外部的

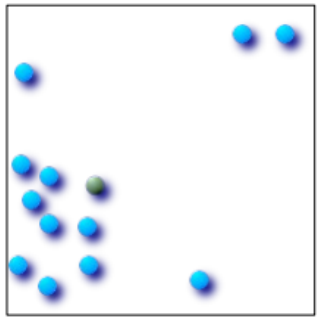
2) 环坐标必须闭合

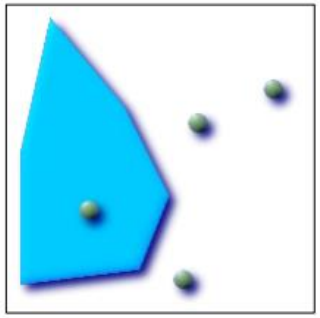
# 几何：多重和聚合Multiples and aggregates

多点

多线

多面





几何集合

1） 不同的投影系统不能是混合的

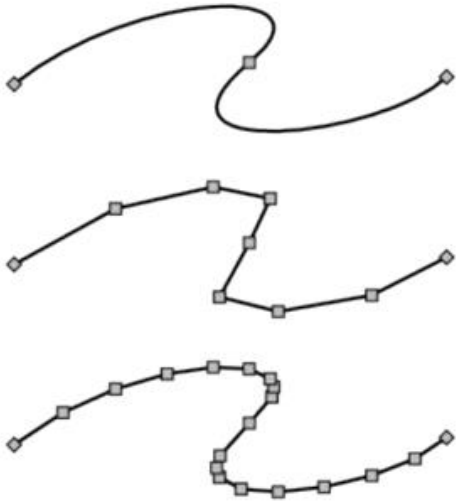
2） 不同的维度也不能

# 几何：曲线curves

圆弧CIRCULARSTRING：

圆弧CIRCULARSTRING是基本曲线类型，类似于线性世界中的线串。单个线段需要三个点，起点和终点（第一个和第三个）以及圆弧上的任何其他点。

循环串（0 0，4 0，4 4，0 4，0 0）

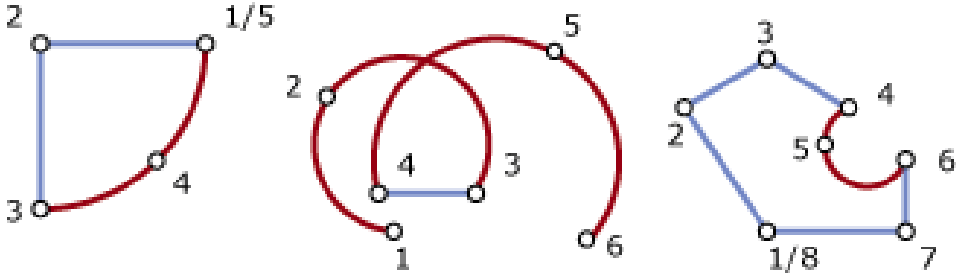


# 几何：曲线COMPOUNDCURVE

复合曲线COMPOUNDCURVE：

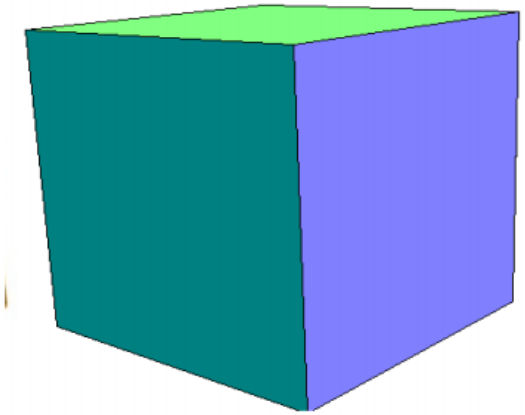
复合曲线是一条既有曲线（圆形）段又有直线段的单一连续曲线。这意味着，除了具有良好的组件外，每个组件的端点（除了最后一个）必须与以下组件的起点重合。

COMPOUNDCURVE(CIRCULARSTRING(0 0, 1 1, 1 0),(1 0, 0 1))



# 几何：多面体表面（PostGIS 2.0）

```
PolyhedralSurface(((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)),
                  ((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)),
                  ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),
                  ((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),
                  ((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)),
                  ((0 0 1, 1 0 1, 1 1 1, 0 1 1. 0 0 1)))
```



# 空间数据特点

慕课视频片段

视频名称：空间数据的特点

温馨提示：此视频框在点击“上传手机课件”时会进行转换，用手机进行观看时则会变为可点击的视频。此视频框可被拖动移位和修改大小

# 数据库中的PostGIS：附加表

## geometry_columns : spatial fields catalog

| | oid | f_table_catalog [PK] character va | f_table_schema [PK] character v | f_table_name [PK] character v | f_geometry_column [PK] character varyi | coord_dimension integer | srid integer | type character varying(30) |
|---|---|---|---|---|---|---|---|---|
| 1 | 709958 | '' | public | dept | the_geom | 2 | 27582 | MULTIPOLYGON |
| 2 | 709957 | '' | public | world | the_geom | 2 | 4326 | MULTIPOLYGON |

## spatial_ref_sys: projection systems catalog

| | srid [PK] integer | auth_name character var | auth_srid integer | srtext character varying(2048) | proj4text character varying(2048) |
|---|---|---|---|---|---|
| 1 | 2000 | EPSG | 2000 | PROJCS["Anguilla 1957 / British We | +proj=tmerc +lat_0=0 +lon_0=-62 +k=0.999 |
| 2 | 2001 | EPSG | 2001 | PROJCS["Antigua 1943 / British We | +proj=tmerc +lat_0=0 +lon_0=-62 +k=0.999 |
| 3 | 2002 | EPSG | 2002 | PROJCS["Dominica 1945 / British W | +proj=tmerc +lat_0=0 +lon_0=-62 +k=0.999 |

Hint : can be interesting to store these tables in a different schema

# 索引

索引用于加速查询和快速定位行

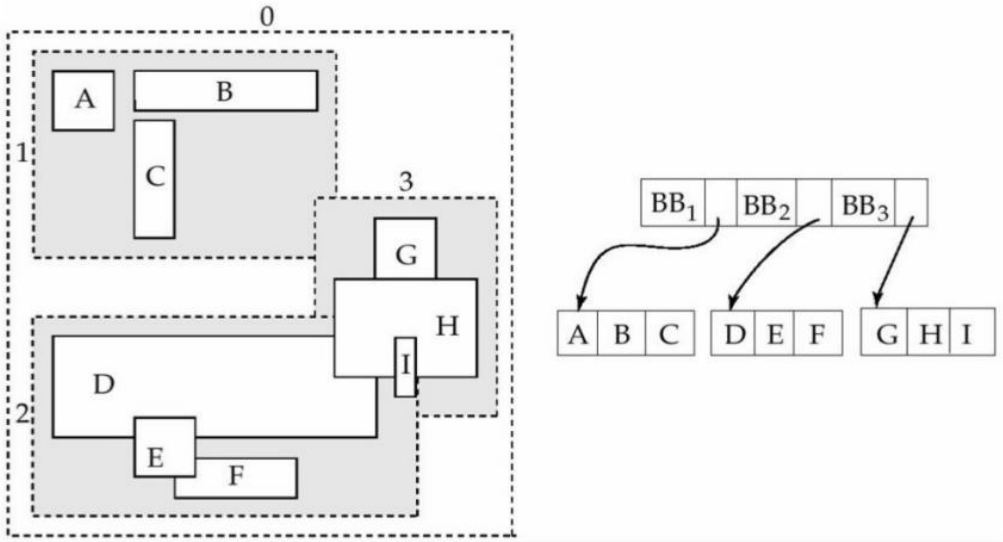
传统 RDBMS 使用一维索引（B-tree）

空间 DBMS 需要二维，分层索引

- 网格
- 四叉树
- R-tree
- 其他

数据库对多维数据的存取有两种索引方案，R-Tree 和 GiST (Generalized Search Tree)，在 PostgreSQL 中的 GiST 比 R-Tree 的健壮性更好，因此 PostGIS 对空间数据的索引一般采用 GiST 实现。

# R-tree

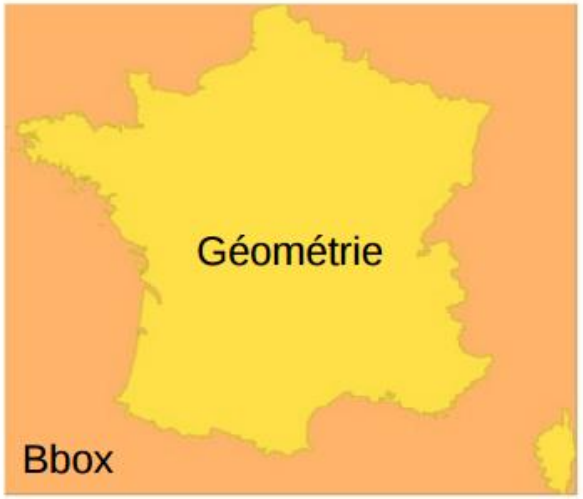



边界框按索引区域分组

# 创建空间索引

更好的空间滤波器性能

带Bbox的几何近似

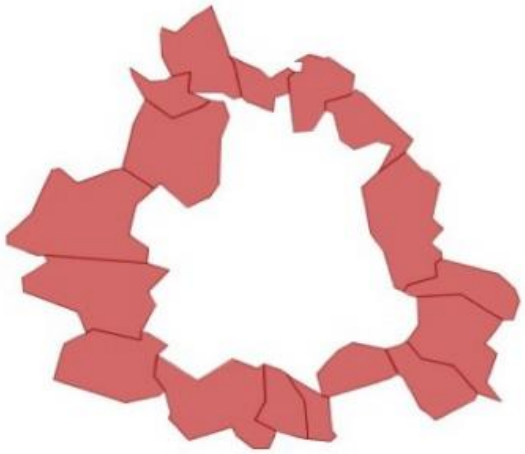



创建空间索引：

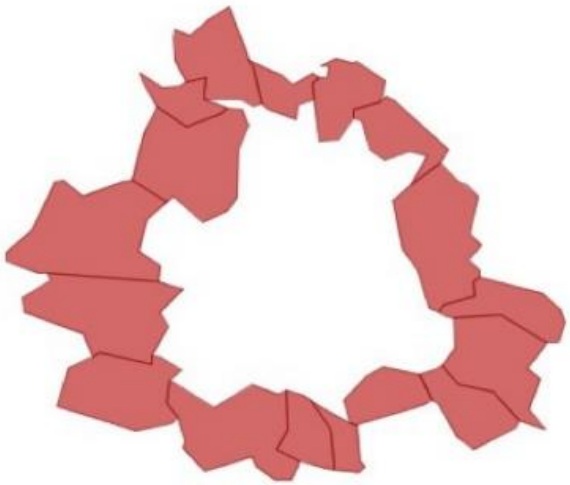
```
CREATE INDEX index_name ON table_name
USING GIST(geom_column_name);
```

# 空间索引

```
SELECT c1.nom FROM communes c1, communes c2
WHERE c2.nom = 'Toulouse'
AND ST_Touches(c1.the_geom, c2.the_geom);
```



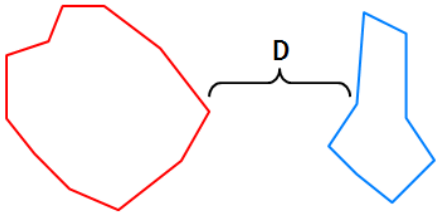
无索引：时间=150ms



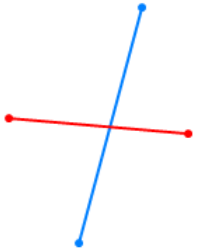
索引：时间=30ms

# PostGIS中的空间操作与关系

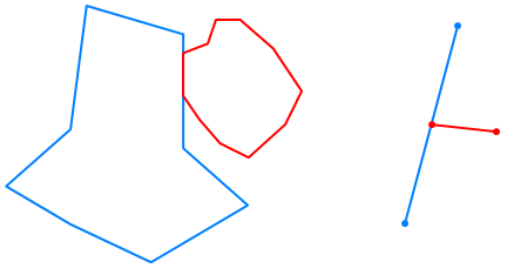
ST_DWithin(A, B, D)



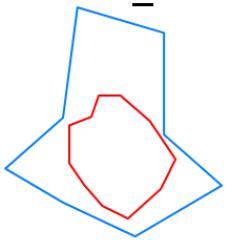

ST_Crosses(A, B)



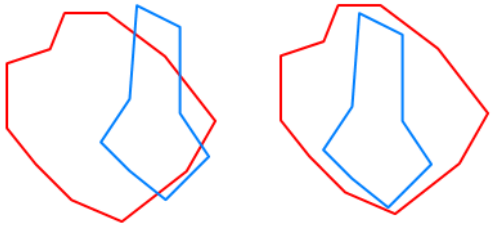
ST_Touches(A, B)



ST_Contains(A, B)
ST_Within(B, A)



ST_Intersects(A, B)



# 单选题 1分

设置

现有一街道几何的文本表示为LINESTRING(586782 4504202,586864 4504216)，SRID为26918，请填充以下SQL语句，返回街区表中与该街道存在公共部分的街区的名称及行政区名称。从文本表示得到几何的函数为ST_GeomFromText(text,srid)

```
SELECT name,boroname
FROM nyc_neighborhoods
WHERE [填空1] (geom, ST_GeomFromText('LINESTRING(586782 4504202,586864 4504216)',26918))
```

A ST_Contains

B ST_Touches

C ST_Within

D ST_Intersects

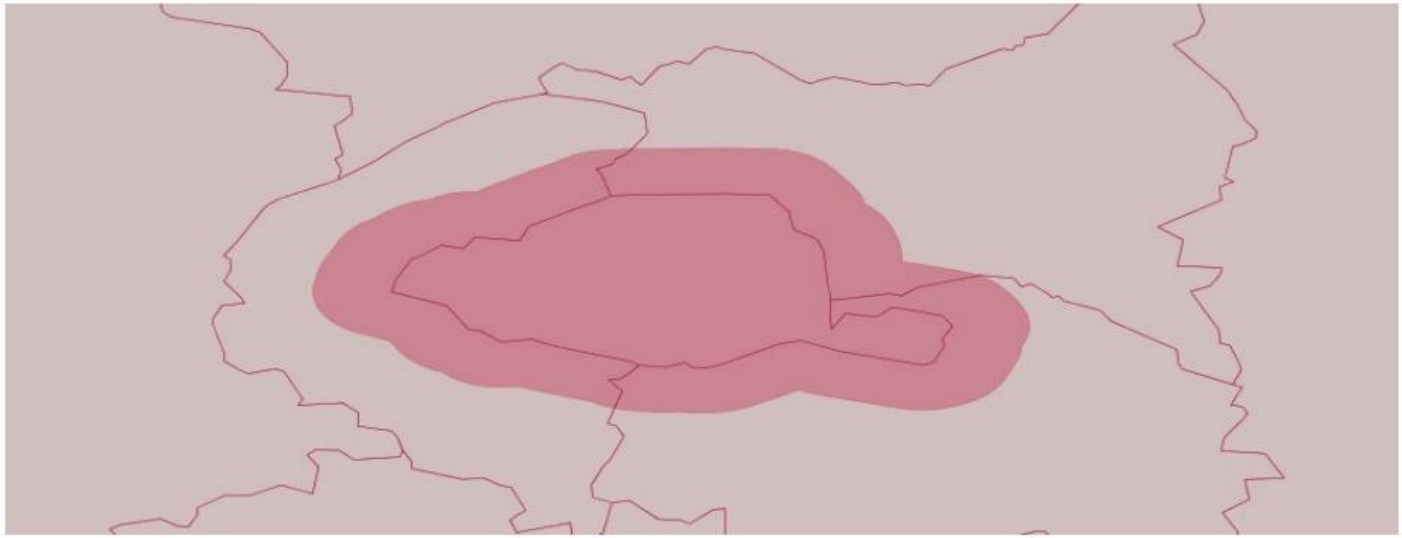
nyc_neighborhoods（街区）表

| name（街区名称） | boroname（行政区名称） | geom（几何表示） |
|---|---|---|
| Bensonhurst | Brooklyn | "0106000000010..." |
| East Village | Manhattan | "0106000000010..." |
| West Village | Manhattan | "0106000000010..." |
| Throggs Neck | The Bronx | "0106000000010..." |
| Wakefield-Williamsbridge | The Bronx | "0106000000010..." |

提交

# PostGIS函数：缓冲区

```
SELECT ST_Buffer(the_geom, 2500)
FROM dept
WHERE code_dept='75';
```



# PostGIS函数：几何图形聚合

Les communes de France

```
SELECT ST_Union(the_geom)
FROM commune
GROUP BY code_dept;
```

# 在聚合查询中使用空间操作

查询：列出所有国家，按接壤国家的数量排序。

```
SELECT Co.Name, Count(Co1.Name)
FROM Country Co, Country Co1
WHERE Touch(Co.Shape,Co1.Shape)
GROUP BY Co.Name
ORDER BY Count(Co1.Name)
```

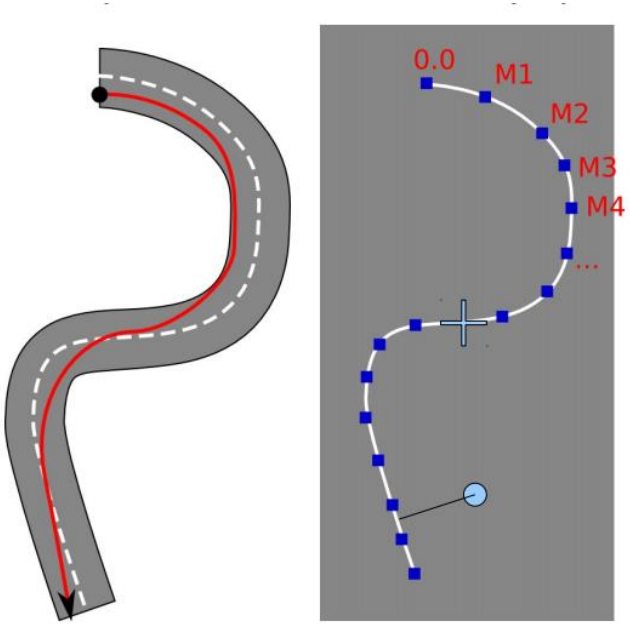
注意：此查询可用于区分简单GIS软件（例如Arc/View）和空间数据库的查询功能。在GIS中进行这种查询是相当繁琐的。

OGIS的早期版本没有提供空间聚合操作来支持重分类等GIS操作。

# PostGIS函数：线性参照

**线性参照函数**

(以道路网为例)




根据location（0-1）获得该位置的点

ST_line_interpolate_point(linestring, location)

获取一段线

ST_line_substring(linestring, start, end)

根据点到线的最近距离获取location（0-1）

ST_line_locate_point(LineString, Point)

# PostGIS功能：pgRouting

pgRouting，一个额外的图形路径模块


http://www.pgrouting.org/

# 示例：地图匹配

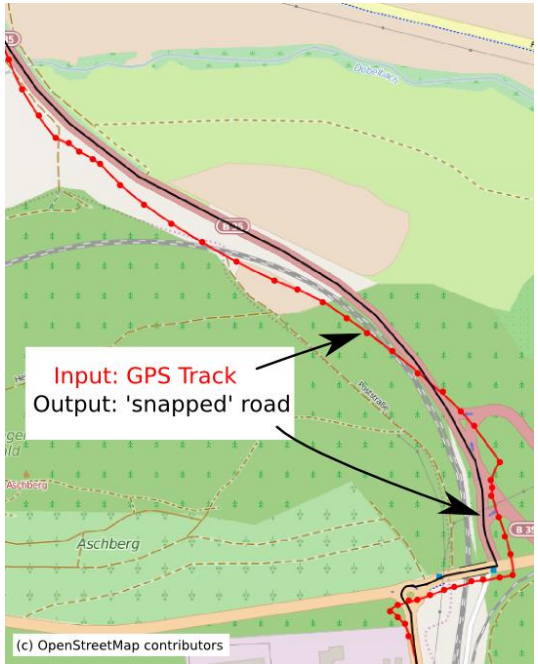
地图匹配

匹配道路网中路径的轨迹

- 使用车队生成的浮动车数据（FCD）作为样本评估总体交通状况。
- 浮动车数据（FCD）

  基本车辆遥测，例如速度、方向、ABS使用

  GPS跟踪获得的车辆位置（→追踪数据）



# PostGIS中的地图匹配

```
SELECT ST_Line_Locate_Point(st_linemerge(line.geom),gps.geom)
from (select * from map where gid=3) as line, gps
```

运行该语句，可以在数据库中输出所有原始GPS数据点匹配后的匹配点在地图路段中的准确位置（用路段百分比表示）

| | st_line_locate_point double precision |
|---|---|
| 1 | 0.431607617353153 |
| 2 | 0.232125185702868 |
| 3 | 0.434798193376835 |
| 4 | 0.433911864762634 |
| 5 | 0.129630224435189 |
| 6 | 0.443521841845728 |
| 7 | 0.426350681836046 |
| 8 | 0.445792393715315 |
| 9 | 0.172892246474788 |
| 10 | 0.436960514030406 |
| 11 | 0.196926522984398 |
| 12 | 0.447031745739292 |
| 13 | 0.172567076867692 |
| 14 | 0.445396083509846 |
| 15 | 0.43336842746733 |
| 16 | 0.433739875919728 |
| 17 | 0.443867797493147 |
| 18 | 0.445921031363585 |
| 19 | 0.226201902331136 |
| 20 | 0.445131052811865 |

图 6.4 地图匹配点在道路上的百分比位置

# PostGIS中的地图匹配

```
SELECT ST_Line_Interpolate_Point(st_linemerge(line.geom),
ST_Line_Locate_Point(st_linemerge(line.geom),gps.geom))
from (select * from map where gid=3) as line, gps
```

通过以上的查询语句，可以获得匹配点在道路上的准确位置，接下来就是将百分比形式的位置信息转化为与原始数据数据类型相同的经纬度坐标。

| | st_astext<br>text |
|---|---|
| 1 | POINT(113.280118517272 23.3538383817599) |
| 2 | POINT(113.290718402638 23.3625634409027) |
| 3 | POINT(113.27993200731 23.3537206117059) |
| 4 | POINT(113.279973 23.353768001) |
| 5 | POINT(113.29262640809 23.369520868119) |
| 6 | POINT(113.279431055803 23.3533718929765) |
| 7 | POINT(113.280457845041 23.3539899139044) |
| 8 | POINT(113.279297871936 23.3532822959517) |
| 9 | POINT(113.291843772555 23.3665941231118) |
| 10 | POINT(113.279832 23.353604999) |
| 11 | POINT(113.291404730414 23.3649527510059) |
| 12 | POINT(113.279225175206 23.3532333905487) |
| 13 | POINT(113.291849864165 23.3666162891585) |
| 14 | POINT(113.279321118322 23.3532979345353) |
| 15 | POINT(113.280003001 23.3537919990001) |
| 16 | POINT(113.279982494816 23.3537755959669) |
| 17 | POINT(113.279410763066 23.353358241407) |
| 18 | POINT(113.279290326431 23.3532772198507) |
| 19 | POINT(113.290872 23.3629530000001) |
| 20 | POINT(113.27933666424 23.353308392769) |

图 6.5 匹配点经纬度坐标输出

# 常见的PostGIS函数

几何对象存取函数：

- 获取几何对象的WKT描述 ST_AsText(geometry)
- 获取几何对象的WKB描述 ST_AsBinary(geometry)
- 获取几何对象的空间参考ID ST_SRID(geometry)
- 获取点的X坐标 ST_X(geometry)
- 获取点的Y坐标 ST_Y(geometry)

几何对象处理函数：

- 获取两个几何对象间的距离 ST_Distance(geometry, geometry)
- 判断两个几何对象是否相交 ST_Intersects(geometry, geometry)
- 判断两个几何对象的边缘是否接触 ST_Touches(geometry, geometry)
- 获取几何对象的中心 ST_Centroid(geometry)
- 面积测量 ST_Area(geometry)
- 长度测量 ST_Length(geometry)
- 获取缓冲后的几何对象 ST_Buffer(geometry, double, [integer])

# 常见的PostGIS函数

几何操作符：：

- A二维相交B A&&B
- A包含B A~B
- A边界在B左边 A<<B
- A边界在B右边 A>>B
- A边界被B包含 A@B

几何量测函数：

- ST_Distance_Sphere(point, point)：根据经纬度点计算在地球曲面上的距离，单位米，地球半径取值6370986米
- ST_Distance_Spheroid(point, point):两个经纬度坐标之间距离，精度较高，效率比ST_Distance_Sphere稍低
- ST_Distance:根据空间参考计算距离，返回的单位根据投影单位
- ST_max_distance(linestring,linestring):量测两条线之间的最大距离
- ST_azimuth(geometry, geometry):量测两点构成的方位角，单位弧度

# 单选题 1分

设置

填充以下SQL语句，使其返回街区表中街区中心点不在街区多边形内部的街区数量。

```
SELECT COUNT(*)
FROM nyc_neighborhoods
WHERE NOT [填空1] (geom, [填空2])
```

A
填空1:ST_Within
填空2:ST_Centroid(geom)

B
填空1:ST_Within
填空2:ST_Buffer(geom,1)

C
填空1:ST_Contains
填空2:ST_Centroid(geom)

D
填空1:ST_Contains
填空2:ST_Length(geom)

nyc_neighborhoods（街区）表

| name（街区名称） | boroname（行政区名称） | geom（几何表示） |
|---|---|---|
| Bensonhurst | Brooklyn | "0106000000010..." |
| East Village | Manhattan | "0106000000010..." |
| West Village | Manhattan | "0106000000010..." |
| Throggs Neck | The Bronx | "0106000000010..." |
| Wakefield-Williamsbridge | The Bronx | "0106000000010..." |

提交

# PostGIS基础函数

基础查询语句：

```
SELECT * FROM china LIMIT 5;
--SELECT geom FROM china WHERE china."name"='东城区';
--SELECT geom FROM china WHERE china."name"='西城区';
--SELECT * FROM china WHERE china."name"='朝阳区';
--SELECT geom FROM china WHERE china."name"='石景山区';
--SELECT * FROM china WHERE china."name"='通州区';
--SELECT geom FROM china WHERE china."name"='海淀区';
--SELECT geom FROM china WHERE china."name"='丰台区';
--SELECT geom FROM china WHERE china."name"='大兴区';
--SELECT geom FROM china WHERE china."name"='房山区';
--SELECT geom FROM china WHERE china."name"='门头沟区';
```

https://www.postgis.net/docs/

# PostGIS基础函数

## 是否相连

```
--SELECT st_disjoint(a.geom, b.geom) from china a, china b  where a.name='石景山区' and b."name"='丰台区';
```

## 是否相交

```
--SELECT st_intersects(a.geom, b.geom) FROM china a,china b where a.name='海淀区' and b."name"='丰台区';
```

## 求距离

```
--SELECT st_distance(a.geom, b.geom) FROM china a,china b where a.name='海淀区' and b."name"='朝阳区';
```

## 是否距离包含

```
--SELECT st_dwithin(a.geom, b.geom,2) FROM china a,china b where a.name='海淀区' and b."name"='朝阳区';
```

# PostGIS基础函数

是否接触

```
--SELECT st_touches(a.geom, b.geom) FROM china a,china b where a.name='海淀区' and b."name"='石景山区';
```

是否重叠

```
--SELECT st_overlaps(a.geom, b.geom) FROM china a,china b where a.name='丰台区' and b."name"='海淀区';
```

求面积

```
--Geometry Accessors查询 --SELECT st_area(geom) FROM china where china.name='丰台区' ;--
```

求长度

```
--SELECT st_length(geom) FROM china WHERE china.name='海淀区';
```

# PostGIS基础函数

求线上的点数

```
--SELECT st_numpoints(geom) FROM china where china.name='海淀区';
```

判断几何类型

```
--SELECT st_geometrytype(geom) FROM china where china.name='漠河县';
```

几何空间数据转换成空间数据文本格式

```
--SELECT st_astext(geom) FROM china WHERE china.name='海淀区';
```

返回当前几何空间数据的SRID值

```
SELECT st_srid(geom) FROM china WHERE china.name='海淀区';
```

# PostGIS基础函数

判断是否闭合

```
--SELECT st_isclosed(geom) FROM china where china.name='海淀区';--
```

判断是否为空

```
--SELECT st_isempty(geom) FROM china where china.name='海淀区';--
```

判断起始点和终点坐标是否相同

```
--SELECT st_isring(geom) FROM china where china.name='海淀区';
```

判断几何对象是否不包含特殊点（比如自相交）

```
--SELECT st_issimple(geom) FROM china where china.name='海淀区';
```

# PostGIS基础函数

## 坐标转换

```
select ST_Transform(geom,4326); //修改表中geom为4517的几何数据修改为4490的数据 update tempjbnt set geom=
st_transform(st_setsrid(geom,4517),4490)
```

## 大地坐标系面积计算

WGS84(4326)是大地坐标系，单位是度（角度单位），角度用来测量长度和面积是不合适的

```
select st_area(geom,true); ---与平面坐标的面积有误差 SELECT
st_area(ST_Transform(st_geometryfromtext('POLYGON((116.4679312706 39.9482801227,116.4677961543
39.9486461337,116.4680989087 39.9486998528,116.4682182670 39.9483181633,116.4679312706
39.9482801227))',4326),4517));------与平面坐标的面积相等
```

国家2000(4527) 是投影坐标系，单位是米，可以计算距离

## 判断点是在哪个多边形里

```
select * from osm_buildings where ST_Within(st_geomfromtext('point(103.76902131950 36.07270404286)',4326),geom);
```

## 获取几何对象的中心点

```
select st_astext( ST_Centroid(ST_GeomFromText('MULTIPOLYGON(((116.3822484 39.9032743,110.3822732
39.9034939,110.3824074 36.9036869,110.3824074 36.9036869,116.3822484 39.9032743)))',4326)) )
```

## 计算指定范围的缓冲区

```
//获取geom 1000米的缓冲范围 select st_buffer ( geom :: geography, 1000 ) from wp_dktb
```