PostgreGIS 及其应用简介

（课程补充阅读材料，编整：谢兴生）

1. 演示空间 GIS 数据集（nyc schema）

（1） 人口普查区块 **nyc_census_blocks** (blkid, popn_total, ….. , boroname, geom)

https://postgis.postgresql.ac.cn/workshops/postgis-intro/simple_sql.html

人口普查区块是报告人口普查数据的最小地理区域。所有更高层级的人口普查地理区域（区块组、区、都市区、县等）都可以通过人口普查区块的合并构建。记录数量：38794

blkid	一个 15 位代码，唯一标识每个人口普查 **区块**。例如：360050001009000
popn_total	人口普查区块中的人口总数
popn_white	在该区块中自我识别为“白人”的人数
popn_black	在该区块中自我识别为“黑人”的人数
popn_nativ	在该区块中自我识别为“美洲原住民”的人数
popn_asian	在该区块中自我识别为“亚洲人”的人数
popn_other	在该区块中自我识别为其他类别的人数
boroname	纽约市区的名称。曼哈顿、布朗克斯、布鲁克林、史坦顿岛、皇后区
geom	区块的**多边形边界**

（2）街区 **nyc_neighborhoods**(name, boroname, geom)

纽约拥有丰富的街区名称和范围历史。街区是社会结构，不遵循政府划定的界线。例如，布鲁克林的卡罗尔花园、红钩和鹅卵石山街区曾经统称为“南布鲁克林”。记录数量：129

name	街区名称
boroname	纽约市区的名称。曼哈顿、布朗克斯、布鲁克林、史坦顿岛、皇后区
geom	街区的**多边形边界**

（3）街道 **nyc_streets** (name, type, geom)

街道中心线构成了城市的交通网络。这些街道已标注类型，以便区分后巷、主干道、高速公路和较小的街道等道路。理想的居住区域可能位于住宅街道上，而不是高速公路旁。记录数量：19091

name	街道名称
oneway	街道是否单行道？“yes” = 是，“” = 否
type	道路类型（主干道、次干道、住宅区、高速公路）
geom	街道的**线性中心线**

（4）地铁站 **nyc_subway_stations** (name, geom)

地铁站将人们居住的上层世界连接到地下的无形地铁网络。作为公共交通系统的门户，车站位置有助于确定不同的人进入地铁系统的难易程度。记录数量：491

name	车站名称
borough	纽约市区的名称。曼哈顿、布朗克斯、布鲁克林、史坦顿岛、皇后区

	routes	经过该站的地铁线路
	transfers	可以通过该站换乘的线路
	express	快车停靠的车站，“express” = 是，“” = 否
	geom	车站的**点位置**

2. 具有 GIS 特性数据的简单 SQL

- 查：“布鲁克林所有街区的名称，及名称的字母数”

```
SELECT name, char_length(name)  FROM nyc_neighborhoods  WHERE boroname = 'Brooklyn';
```

- 查：“布鲁克林所有街区名称的字母数的平均值和标准差是多少？”

```
SELECT avg(char_length(name)), stddev(char_length(name))
                              FROM nyc_neighborhoods  WHERE boroname = 'Brooklyn';
        avg       |      stddev
--------------------+-----------------------------------
11.7391304347826087 | 3.9105613559407395
```

- “纽约市所有街区的名称平均有多少个字母？按区统计。”

```
SELECT boroname, avg(char_length(name)), stddev(char_length(name))
    FROM nyc_neighborhoods   GROUP BY boroname;
```

分组统计允许输出的列：（a）分组子句的成员，（b）各种聚合函数。

```
  boroname     |       avg                    |      stddev
-------------------+------------------------------+--------------------
 Brooklyn          | 11.7391304347826087  | 3.9105613559407395
 Manhattan      | 11.8214285714285714  | 4.3123729948325257
 The Bronx       | 12.0416666666666667  | 3.6651017740975152
 Queens           | 11.6666666666666667  | 5.0057438272815975
 Staten Island | 12.2916666666666667  | 5.2043390480959474
```

- 对于每个区，白人人口占总人口的百分比是多少？

```
SELECT  boroname,  100.0 * Sum(popn_white)/Sum(popn_total) AS white_pct
                                    FROM nyc_census_blocks  GROUP BY boroname;
  boroname     |    white_pct
-------------------+-----------------
 Brooklyn          | 42.8011737932687
 Manhattan      | 57.4493039480463
 The Bronx       |  27.9037446899448
 Queens           |  39.722077394591
 Staten Island |  72.8942034860154
```

3. 空间数据的主要类型及相关的简单查询应用

如何表示现实世界中的对象？PostGIS 开发的最初指导标准是 SQL 简单要素规范(SFSQL)，仅处理二维表示。PostGIS 现已将此扩展到包括三维和四维表示。新版的 SQL-Multimedia 第 3 部分 (SQL/MM) 规范，已正式定义了自己的表示方式。

3.1 几何图形

增补的特别实验数据:

```
CREATE TABLE geometries (name varchar,  geom geometry);

INSERT INTO geometries VALUES
    ('Point', 'POINT(0 0)'),
    ('Linestring', 'LINESTRING(0 0, 1 1, 2 1, 2 2)'),
    ('Polygon', 'POLYGON((0 0, 1 0, 1 1, 0 1, 0 0))'),
    ('PolygonWithHole', 'POLYGON((0 0, 10 0, 10 10, 0 10, 0 0),(1 1, 1 2, 2 2, 2 1, 1 1))'),
    ('Collection', 'GEOMETRYCOLLECTION(POINT(2 0),POLYGON((0 0, 1 0, 1 1, 0 1, 0 0)))');
```

插入了五个几何图形：一个点、一条线、一个多边形、一个带孔的多边形和一个集合。

➢ **SELECT** name, ST_AsText(geom) **FROM** geometries;

name character varying	st_astext text
Point	POINT(0 0)
Linestring	LINESTRING(0 0,1 1,2 1,2 2)
Polygon	POLYGON((0 0,1 0,1 1,0 1,0 0))
PolygonWithHole	POLYGON((0 0,10 0,10 10,0 10,0 0),(1 1,1 2,2 2,2 1,1 1))
Collection	GEOMETRYCOLLECTION(POINT(2 0),POLYGON((0 0,1 0,1 1,0 1,0 0)))

这个示例表包含几个不同几何类型，可使用读取几何元数据的函数，获取对象的常规信息。

- **ST_GeometryType(geometry)** 返回几何类型
- **ST_NDims(geometry)** 返回几何的维数
- **ST_SRID(geometry)** 返回几何的空间参考标识符号

```
SELECT name, ST_GeometryType(geom), ST_NDims(geom), ST_SRID(geom)   FROM geometries;
⇨      name              |   st_geometrytype          | st_ndims  |  st_srid
      ------------------------+-------------------------------+-----------+---------
       Point                | ST_Point                   |     2     |    0
       Polygon              | ST_Polygon                 |     2     |    0
       PolygonWithHole | ST_Polygon                 |     2     |    0
       Collection         | ST_GeometryCollection |     2     |    0
       Linestring         | ST_LineString              |     2     |    0
```

3.2 空间点

- 用于表示孤立对象，例如，世界地图上的城市，或城市中的地铁站，都可以描述为点。
- 由单个坐标表示（包括二维、三维或四维）。
- 一些用于处理点的空间函数:
  - ST_X(geometry) 返回 X 坐标
  - ST_Y(geometry) 返回 Y 坐标

➢ **SELECT** ST_**AsText**(geom)， ST_X(geom), ST_Y(geom) **FROM** geometries **WHERE** name = 'Point';

⇨ *POINT(0 0)，0,0*

➢ 查询纽约市地铁站 (nyc_subway_stations) 表将返回与之关联的几何图形维度（在 ST_AsText 列中）。

```
SELECT name, ST_AsText(geom)  FROM nyc_subway_stations  LIMIT 1
```

⇨ *POINT;*

3.3 线串

- 线串是位置之间的路径，是两个或多个点的有序序列的形式。
- 道路和河流通常表示为线串。如果线串的起点和终点位于同一点，则称其为闭合。
- 如果线串不交叉或不与自身接触（除非在闭合时在端点处），则称其为简单。
- 线串可以同时是闭合和简单的。
- 一些用于处理线串的空间函数：
  - ST_Length(geometry)　返回线串的长度
  - ST_StartPoint(geometry) 返回第一个坐标作为点
  - ST_EndPoint(geometry) 返回最后一个坐标作为点
  - ST_NPoints(geometry)　返回线串中的坐标数量

➢ 查询线串的长度：

```
SELECT ST_Length(geom)  FROM geometries   WHERE  name = 'Linestring';
```

⇨ *3.41421356237309*

3.4 多边形

- 多边形的外部边界由一条闭环线串表示，它既是闭合的又是简单的。
- 当比例尺足够高俯瞰区域，城市界限、公园、建筑占地面积或水体通常都以多边形表示。道路和河流有时也可以用多边形表示。
- 一些用于处理多边形的空间函数：
  - ST_Area(geometry) 返回多边形的面积
  - ST_NRings(geometry) 返回环的数量（通常为 1，如果存在孔则更多）
  - ST_ExteriorRing(geometry) 返回外环作为线字符串
  - ST_InteriorRingN(geometry,n) 返回指定的内环作为线字符串
  - ST_Perimeter(geometry) 返回所有环的长度

➢ 查询 与多边形相关的几何图形（在 ST_AsText 列中）。

```
SELECT ST_AsText(geom)  FROM geometries   WHERE name LIKE 'Polygon%';
```

⇨ *POLYGON((0 0, 1 0, 1 1, 0 1, 0 0))*

⇨ *POLYGON((0 0, 10 0, 10 10, 0 10, 0 0),(1 1, 1 2, 2 2, 2 1, 1 1))*

第一个多边形只有一个环。第二个多边形有一个内部“孔”。

➢ 使用面积函数计算多边形的面积

```
SELECT name, ST_Area(geom)  FROM geometries   WHERE  name LIKE 'Polygon%';
```

⇨ *Polygon　　1*

⇨ *PolygonWithHole　99*

**带孔的多边形的面积是外壳面积（10x10 平方）减去孔面积（1x1 平方）。*

➢ “西村”街区的面积是多少？

SELECT ST_Area(geom) **FROM** nyc_neighborhoods **WHERE** name = 'West Village';

⇨ *1044614.5296486*

面积以平方米为单位。/10000 =>公顷; /4047==>英亩面积。

➢ 曼哈顿的面积是多少英亩？

SELECT Sum(ST_Area(geom)) / 4047 **FROM** nyc_neighborhoods **WHERE** boroname = 'Manhattan';

⇨ *13965.3201224118*

3.5 集合

- 集合在 GIS 软件中比在通用图形软件中出现得更多。它们对于直接将现实世界中的对象建模为空间对象，并进行统计很有用。
- PG-GIS 中有四种集合类型，它们可将多个简单几何图形分组为集合。
  - MultiPoint， 点集合
  - MultiLineString，线串集合
  - MultiPolygon， 多边形集合
  - GeometryCollection，任何几何图形（包括其他集合）的异构集合
- 用于处理集合的一些特定空间函数：
  - ST_NumGeometries(geometry) 返回集合中部件的数量
  - ST_GeometryN(geometry,n) 返回指定的部件
  - ST_Area(geometry) 返回所有多边形部件的总面积
  - ST_Length(geometry) 返回所有线性部件的总长度

➢ 查询集合（*结果一条，包含一个多边形和一个点*）

SELECT name, ST_AsText(geom) **FROM** geometries **WHERE name = 'Collection'**;

⇨ *GEOMETRYCOLLECTION(POINT(2 0),POLYGON((0 0, 1 0, 1 1, 0 1, 0 0)))*

3.6 其他 GIS 小应用示例

➢ “Pelham St”的几何类型是什么？长度是多少？

SELECT ST_GeometryType(geom), ST_Length(geom) **FROM** nyc_streets **WHERE** name = 'Pelham St';

⇨ *ST_MultiLineString， 50.323*

➢ “Broad St”地铁站的 GeoJSON 表示是什么？

SELECT ST_AsGeoJSON(geom) **FROM** nyc_subway_stations **WHERE** name = 'Broad St';

⇨ *{"type":"Point",*
"crs":{"type":"name","properties":{"name":"EPSG:26918"}},
"coordinates":[583571.905921312,4506714.341192182]}

➢ 纽约市街道的总长度（以公里为单位）是多少？**（单位是米）

SELECT Sum(ST_Length(geom)) / 1000 **FROM** nyc_streets;

⇨ *10418.9047172*

➢ 最西边的地铁站是哪个？

```
SELECT ST_X(geom), name  FROM nyc_subway_stations  ORDER BY ST_X(geom)  LIMIT 1;
```

⇨ *Tottenville*

➢ 纽约市街道的长度，按类型汇总？

```
SELECT type, Sum(ST_Length(geom)) AS length  FROM nyc_streets
        GROUP BY type  ORDER BY length DESC;
```

```
          type          |     length
------------------------------------------------+------------------
residential                | 8629870.33786606
motorway                   | 403622.478126363
tertiary                   | 360394.879051303
motorway_link              | 294261.419479668
……
```

4. 空间关系

以上我们使用了测量(**ST_Area**、**ST_Length**)、序列化(**ST_GeomFromText**)或反序列化(**ST_AsGML**）几何图形的空间函数。这些函数的共同点是它们一次只能处理一个几何图形。

空间数据库之所以强大，是因为它们不仅存储几何图形，而且还能够比较*几何图形之间的关系*。

4.1 OGC 标准定义的一组比较几何图形方法

（1） **ST_Equals(geometry A, geometry B)**，测试两个几何图形的空间相等性。

- 如果两个相同类型的几何图形具有相同的 x,y 坐标值，则 ST_Equals 返回 TRUE。

➢ 1) 从 nyc_subway_stations 表中，检索名未“Broad St”的条目。

```
SELECT name, geom FROM nyc_subway_stations WHERE name = 'Broad St';
```

⇨

```
 name   |   geom
----------+----------------------------------------------------
Broad St |  0101000020266900000EEBD4CF27CF2141BC17D69516315141
```

注意，

✍ *这里点的表示,虽可读性不太好(0101000020266900000EEBD4CF27CF2141BC17D69516315141)，但它是坐标值的精确表示形式。对于像相等性这样的测试，使用精确的坐标是必要的。*

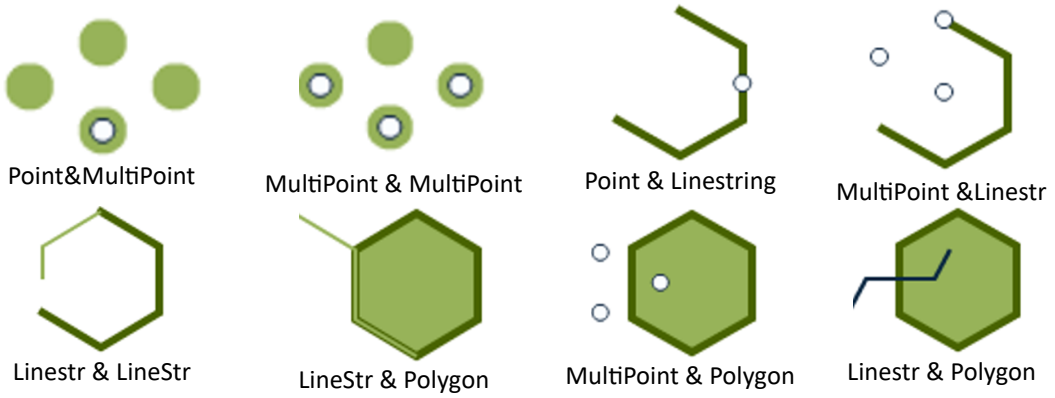
2) 使用 **ST_Equals 作查询**测试

```
SELECT name FROM nyc_subway_stations WHERE ST_Equals(  geom,
        '0101000020266900000EEBD4CF27CF2141BC17D69516315141');
```

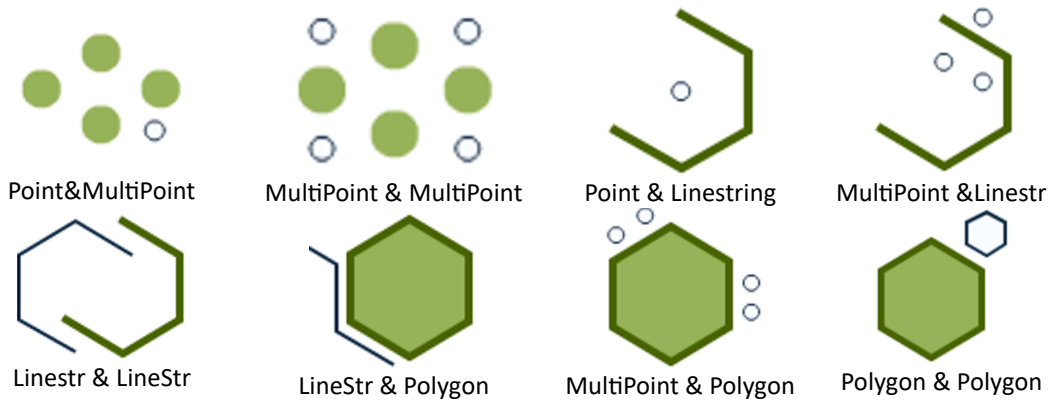
⇨ *Broad St*

（2） 相交性 ST_Intersects、ST_Disjoint、ST_Crosses 和 ST_Overlaps 测试几何图形的内部是否相交

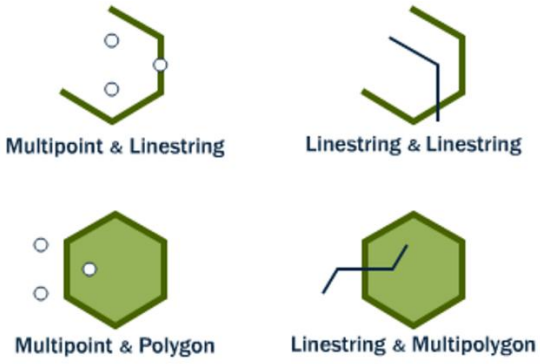
- **ST_Intersects(geometry A, geometry B)**，如果两个形状有任何共同的空间，即如果它们的边界或内部相交，则返回 t（TRUE）。



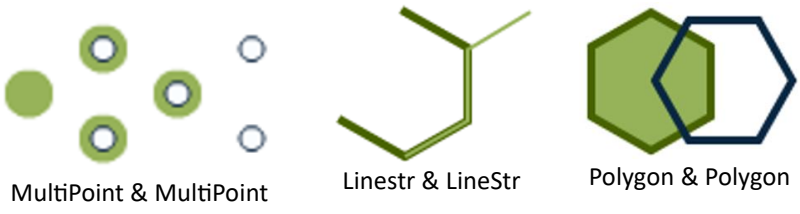
- **ST_Disjoint(geometry A , geometry B)**。若两个几何图形是不相交的，返回 true。测试“相交”通常比测试“不相交”更有效，因为相交测试可以进行空间索引，而不相交测试则不能。



- **ST_Crosses(geometry A, geometry B)** 。 如果交集导致的几何图形的维数比两个源几何图形的最大维数减少，并且交集集位于两个源几何图形的内部，则返回 t（TRUE）。



- **ST_Overlaps(geometry A, geometry B)** 。用于比较两个相同维度的几何图形，如果它们的交集结果是一个与两者不同但维度相同的几何图形，则**返回** TRUE。



相交性应用示例，

- 以百老汇街地铁站为例，使用 ST_Intersects 函数确定其周边区域。

```
SELECT name, ST_AsText(geom)  FROM nyc_subway_stations  WHERE name = 'Broad St';
```

⇨ *POINT(583571 4506714)*

```
SELECT name, boroname FROM nyc_neighborhoods
WHERE ST_Intersects(geom, ST_GeomFromText('POINT(583571 4506714)', 26918) );
```

⇨

```
        name          | boroname
----------------------+------------------
Financial District    | Manhattan
```

- **查询 Atlantic Commons （大西洋公地）位于哪个街区和行政区？**

1) 先查询“Atlantic Commons”的街道的几何结构信息

```
SELECT ST_AsText(geom)   FROM nyc_streets    WHERE name = 'Atlantic Commons';
```

⇨ *MULTILINESTRING((586781.701577724 4504202.15314339,586863.51964484 4504215.9881701))*

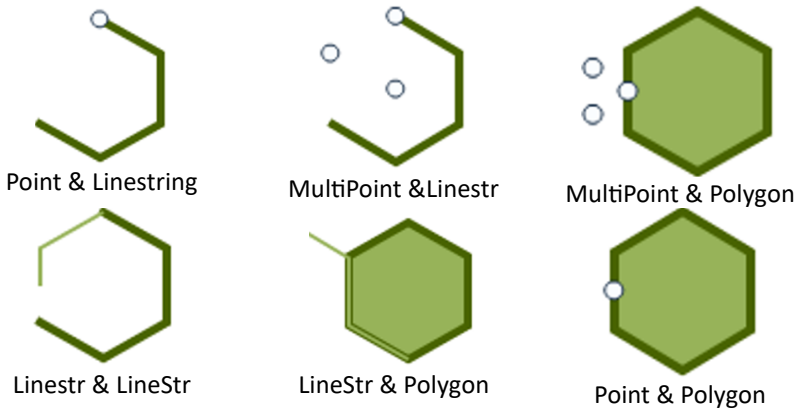
2) 实际执行查询

```
SELECT name, boroname  FROM nyc_neighborhoods
WHERE ST_Intersects(  geom,
                ST_GeomFromText('LINESTRING(586782 4504202,586864 4504216)', 26918) );
```

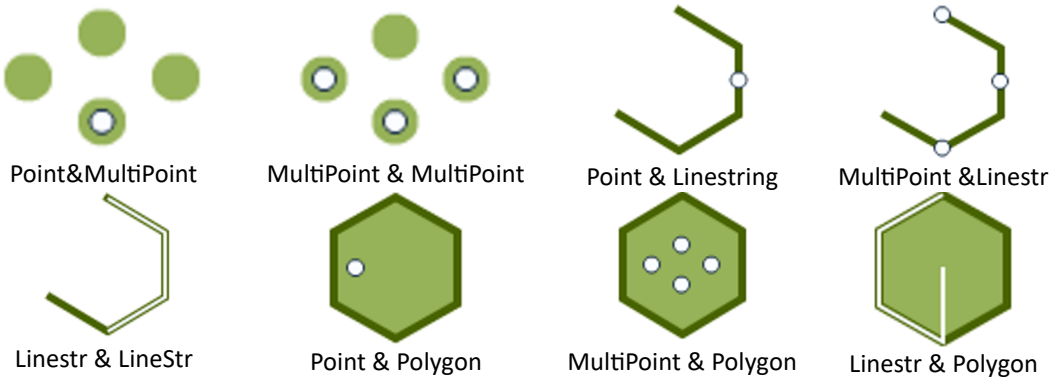
⇨

```
    name      | boroname
--------------+--------------
Fort Green | Brooklyn
```

- **ST_Touches(geometry A, geometry B)** 用于测试两个几何图形是否在边界处接触，但不在内部相交。如果两个几何图形的边界相交，或者只有一个几何图形的内部与另一个几何图形的边界相交，则返回 TRUE



- **ST_Within** 和 **ST_Contains** 用于测试一个几何图形是否完全在另一个几何图形内。
  - **ST_Within(geometry A , geometry B)** 如果第一个几何图形完全在第二个几何图形内，则返回 TRUE。
  - **ST_Contains(geometry A, geometry B)** 测试的结果与 ST_Contains 正好相反。如果第二个几何图形完全包含在第一个几何图形内，则返回 TRUE。

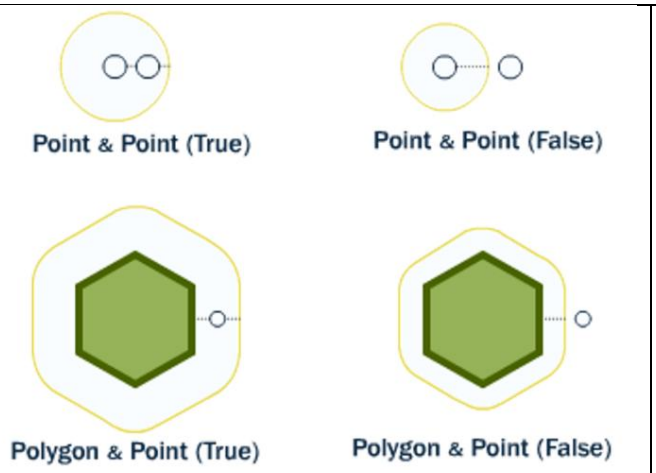


- **ST_Distance (geometry A, geometry B)** 计算两个几何图形之间的**最短**距离，并将其作为浮点数返回。这对于实际报告对象之间的距离很有用。
  - 一个非常常见的 GIS 问题是“找到**距离**某个东西 X 距离内的所有东西”。

```
SELECT ST_Distance(  ST_GeometryFromText('POINT(0 5)'),
                     ST_GeometryFromText('LINESTRING(-2 2, 2 2)'));
```

⇨ *3*

- **ST_DWithin** 。测试两个对象是否在彼此的距离范围内。
  - 该函数提供了一个索引加速的真/假测试。这对于诸如“有多少棵树在道路 500 米缓冲区内？”之类的问题很有用。它可避免计算实际的缓冲区，只需测试距离关系即可。



ST_DWithin 测试语义直观理解

在地图上验证示例查询的答案

➢ 再次用百老汇街地铁站，找到地铁站附近（距离地铁站 10 米以内）的街道。

```
SELECT name   FROM nyc_streets  WHERE ST_DWithin( geom,
                                    ST_GeomFromText('POINT(583571 4506714)',26918),  10 );
```

```
    name
-------------
 Wall St
 Broad St
 Nassau St
```

可以在地图上验证答案：百老汇街车站实际上位于华尔街、百老汇街和纳索街的交汇处。

- 大西洋公地与哪些街道相连？

```
SELECT name  FROM nyc_streets WHERE ST_DWithin(  geom,
        ST_GeomFromText('LINESTRING(586782 4504202,586864 4504216)', 26918),  0.1  );
```

⇨

```
 name
------------------
Cumberland St
Atlantic Commons
```

- 在大西洋公地（50 米范围内）大约住着多少人？

```
SELECT Sum(popn_total)   FROM nyc_census_blocks   WHERE ST_DWithin(   geom,
   ST_GeomFromText('LINESTRING(586782 4504202,586864 4504216)', 26918),  50   );
```

⇨ *1438*

5. 空间连接

空间连接是空间数据库应用的核心。它允许我们通过使用空间关系作为连接键，来组合来自不同表的的信息。实际上，“标准 GIS 分析”中的大部分内容都可以表示为空间连接(用空间连接实现)。

5.1 使用空间连接探索空间关系

在上一节中，我们实际已使用了一个两步过程来探索空间关系。比如，首先，提取了“Broad St”的地铁站点；然后，使用该点来提出进一步的问题，例如“‘Broad St’车站位于哪个街区？”

使用空间连接，我们可以在一步中回答这个问题，检索有关地铁站及其包含的街区的信息。

```
SELECT   subways.name AS subway_name,   neighborhoods.name AS neighborhood_name,
        neighborhoods.boroname AS borough
FROM nyc_neighborhoods AS neighborhoods
JOIN nyc_subway_stations AS subways ON ST_Contains(neighborhoods.geom, subways.geom)
WHERE subways.name = 'Broad St';
```

⇨

```
 subway_name | neighborhood_name  |  borough
-------------+----------------------------+-----------
 Broad St    | Financial District       | Manhattan
```

我们可以将每个地铁站与其包含的街区连接起来，但在这种情况下，我们只想要有关一个的信息。任何在两个表之间提供真/假关系的函数都可以用来驱动空间连接，但最常用的函数是：**ST_Intersects**、**ST_Contains** 和 **ST_DWithin**。

例 1 哪个地铁站在“小意大利”？它在哪个地铁线路？

```
SELECT s.name, s.routes  FROM nyc_subway_stations AS s
      JOIN nyc_neighborhoods AS n  ON ST_Contains(n.geom, s.geom) WHERE n.name = 'Little Italy';
```

⇨

```
 name       | routes
--------------+---------
 Spring St  | 6
```

例 2 6 号线服务哪些街区？

（提示：nyc_subway_stations 表中的 routes 列具有诸如“B,D,6,V”和“C,6”之类的值）

```
SELECT DISTINCT n.name, n.boroname  FROM nyc_subway_stations AS s
      JOIN nyc_neighborhoods AS n  ON ST_Contains(n.geom, s.geom)  WHERE strpos(s.routes,'6') > 0;
```

⇨

```
 name            | boroname
```

```
---------------------+-----------
Midtown              | Manhattan
Hunts Point          | The Bronx
Gramercy             | Manhattan
.....................
```

5.2 连接和汇总

JOIN 与 GROUP BY 的组合提供了通常在 GIS 系统中完成的分析类型。

例如：**“曼哈顿街区的总人口和种族构成如何？”**这里我们有一个问题，它将来自人口普查的人口信息与街区边界的信息结合起来，并限制在曼哈顿的一个区。

```
SELECT   neighborhoods.name AS neighborhood_name,
         Sum(census.popn_total) AS population,
         100.0 * Sum(census.popn_white) / Sum(census.popn_total)  AS white_pct,
         100.0 * Sum(census.popn_black) / Sum(census.popn_total)   AS black_pct
FROM nyc_neighborhoods AS neighborhoods
JOIN nyc_census_blocks AS census ON ST_Intersects(neighborhoods.geom, census.geom)
WHERE neighborhoods.boroname = 'Manhattan'
GROUP BY neighborhoods.name   ORDER BY white_pct DESC;
```

⇨

```
neighborhood_name  | population | white_pct | black_pct
---------------------------+----------------+---------------+-----------
Carnegie Hill          |    18763   |    90.1     |    1.4
North Sutton Area  |    22460   |    87.6     |    1.6
West Village           |    26718   |    87.6     |    2.2
.....
```

在这个查询里：

1) **JOIN 子句创建一个虚拟表，其中包含来自街区表和人口普查表的列。**JOIN 子句将两个 FROM 项目组合在一起。默认情况下，使用的是 INNER JOIN，但还有其他四种类型的联接。
2) **WHERE 子句将我们的虚拟表过滤为仅曼哈顿的行列。**
3) **剩余的行按街区名称分组，并通过聚合函数 Sum()对人口值进行求和。**
4) **在对最终数字进行一些算术运算和格式化（例如，GROUP BY，ORDER BY）后，我们的查询会输出百分比。**

我们还可以使用**距离测试**作为联接键，以创建汇总的“半径内所有项目”查询。让我们使用距离查询来探索纽约的种族地理分布。

➢ 首先，让我们获取该市的种族构成基线。

```
SELECT   100.0 * Sum(popn_white) / Sum(popn_total) AS white_pct,
         100.0 * Sum(popn_black) / Sum(popn_total) AS black_pct,
         Sum(popn_total) AS popn_total
FROM nyc_census_blocks;
```

⇨

```
white_pct    |      black_pct         | popn_total
------------------+---------------------------+------------
44.0039500762811 | 25.5465789002416 |   8175032
```

因此，在纽约的 800 万人口中，大约 44% 被记录为“白人”，26% 被记录为“黑人”。

公爵·艾灵顿曾经唱过“你/必须乘坐 A 线/去/哈莱姆的糖山”。正如我们之前看到的，哈莱姆是曼哈顿非洲裔美国人人口最多的地区（80.5%）。公爵的 A 线也是这样吗？

首先，请注意 nyc_subway_stations 表的 routes 字段的内容是我们感兴趣的，用于查找 A 线。

而要查找 A 线，需要找 routes 中包含“A”的任何行，并用 DISTINCT 从结果中删除重复行。

➢ 我们简单**使用**以下事实：**strpos(routes,'A')** 仅当“A”位于 routes 字段中时才会返回非零数字。

```
SELECT DISTINCT routes FROM nyc_subway_stations AS subways WHERE strpos(subways.routes,'A') > 0;
```

⇨
```
A,B,C
A,C
A
A,C,G
A,C,E,L
A,S
A,C,F
A,B,C,D
A,C,E
```

➢ **查询分析 A 线 200 米范围内的种族构成：**

```
SELECT   100.0 * Sum(popn_white) / Sum(popn_total) AS white_pct,
         100.0 * Sum(popn_black) / Sum(popn_total) AS black_pct,
          Sum(popn_total) AS popn_total
FROM nyc_census_blocks AS census
JOIN nyc_subway_stations AS subways ON ST_DWithin(census.geom, subways.geom, 200)
WHERE strpos(subways.routes,'A') > 0;
```

⇨
```
    white_pct          |   black_pct         | popn_total
---------------------------+-------------------------+------------
45.5901255900202 | 22.0936235670937 |    189824
```

因此，A 线沿线的种族构成与整个纽约市的构成并没有太大区别。

例 3 9/11 事件后，“电池公园”社区被封锁了几天。有多少人被疏散了？

```
SELECT Sum(popn_total)  FROM nyc_neighborhoods AS n
        JOIN nyc_census_blocks AS c  ON ST_Intersects(n.geom, c.geom)
WHERE n.name = 'Battery Park';
```

⇨ *17153*

例 4 哪个社区的人口密度（人/平方公里）最高？

```
SELECT   n.name,   Sum(c.popn_total) / (ST_Area(n.geom) / 1000000.0) AS popn_per_sqkm
FROM nyc_census_blocks AS c
        JOIN nyc_neighborhoods AS n ON ST_Intersects(c.geom, n.geom)
GROUP BY n.name, n.geom ORDER BY popn_per_sqkm DESC LIMIT 2;
```

⇨
```
     name              |  popn_per_sqkm
-------------------------+------------------
North Sutton Area |  68435.13283772678
East Village          | 50404.48341332535
```

5.3 高级联接

在上一节中，我们看到 A 线的种族人口与城市其他地区的种族构成没有太大区别。是否有任何线路的种族构成与平均水平不同？

为了回答这个问题，我们将向查询中添加另一个联接，以便我们可以同时计算许多地铁线路的构成。为此，我们需要创建一个新表，列出我们要汇总的所有线路。

```
CREATE TABLE subway_lines ( route char(1) );
INSERT INTO subway_lines (route) VALUES   ('A'),('B'),('C'),('D'),('E'),('F'),('G'),
                                  ('J'),('L'),('M'),('N'),('Q'),('R'),('S'),   ('Z'),('1'),('2'),('3'),('4'),('5'),('6'),   ('7');
```

现在我们可以将地铁线路表加入到我们的原始查询中。

```
SELECT   lines.route,   100.0 * Sum(popn_white) / Sum(popn_total) AS white_pct,
        100.0 * Sum(popn_black) / Sum(popn_total) AS black_pct,    Sum(popn_total) AS popn_total
FROM nyc_census_blocks AS census
JOIN nyc_subway_stations AS subways ON ST_DWithin(census.geom, subways.geom, 200)
```

```
JOIN subway_lines AS lines ON strpos(subways.routes, lines.route) > 0
GROUP BY lines.route  ORDER BY black_pct DESC;
```

⇨

```
route |white_pct|black_pct| popn_total
------+----------+----------+-----------
S     |    39.8 |    46.5 |     33301
3     |    42.7 |    42.1 |    223047
5     |    33.8 |    41.4 |    218919
2     |    39.3 |    38.4 |    291661
C     |    46.9 |    30.6 |    224411
4     |    37.6 |    27.4 |    174998
B     |    40.0 |    26.9 |    256583
A     |    45.6 |    22.1 |    189824
J     |    37.6 |    21.6 |    132861
.....
```

与之前一样，连接创建了一个虚拟表，其中包含了 JOIN ON 限制条件下所有可能的组合，然后这些行被馈送到 GROUP 汇总中。空间魔法在于 ST_DWithin 函数，它确保只有靠近相应地铁站的普查区被包含在计算中。

6. 空间索引

空间索引是空间数据库的三个关键特征之一。索引使得对大型数据集使用空间数据库成为可能。如果没有索引，任何对要素的搜索都需要对数据库中的每个记录进行“顺序扫描”。索引通过将数据组织成搜索树来加快搜索速度，该搜索树可以快速遍历以查找特定记录。

6.1 为何需要空间索引

空间索引是 PostGIS 的最大优势之一。在之前的示例中，构建空间连接需要将整个表相互比较。这可能会非常昂贵：连接两个各有 10,000 条记录的表，如果没有索引，则需要 100,000,000 次比较；而使用索引，成本可能低至 20,000 次比较。

范例演示：

（1）首先在没有空间索引的情况下，对 nyc_census_blocks 运行查询。

首先，删除已有的表空间索引（如果之前已经建过）

```
DROP INDEX nyc_census_blocks_geom_idx;
```

（2）发出查询，并观察执行情况

```
SELECT count(blocks.blkid)  FROM nyc_census_blocks blocks
  JOIN nyc_subway_stations subways  ON ST_Contains(blocks.geom, subways.geom)
  WHERE subways.name LIKE 'B%';
```

观察 pgAdmin 查询窗口右下角的“计时”仪表，并运行搜索每个普查区块的查询，以识别包含以“B”开头的地铁站的区块。

nyc_census_blocks 表非常小（只有几千条记录），因此即使没有索引，在测试计算机上，查询也只花费 **300 毫秒**。

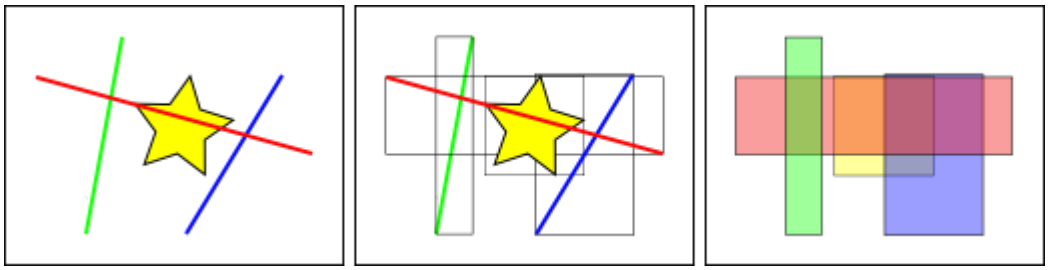
（3）重新添加空间索引，并再次运行查询。

```
CREATE INDEX nyc_census_blocks_geom_idx  ON nyc_census_blocks  USING GIST (geom);
```

USING GIST 子句告诉 PostgreSQL 在构建索引时使用通用索引结构 (GIST)。如果创建索引时收到类似 ERROR: index row requires 11340 bytes, maximum size is 8191 的错误，则很可能忘记添加 USING GIST 子句。在作者的测试计算机上，时间降至 **50 毫秒**。表越大，索引查询的相对速度提升就越大。

6.2 空间索引的工作原理

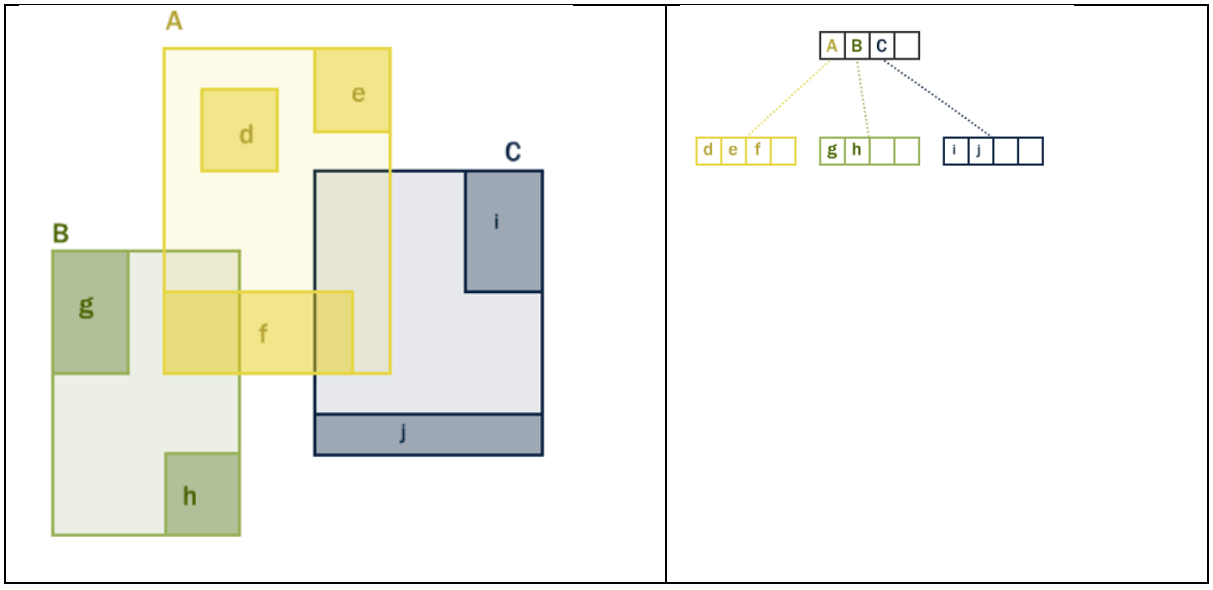
标准数据库索引（B+树）基于被索引的列的值创建一个层次树。空间索引略有不同 - 它们无法索引几何特征本身，而是*索引特征的边界框*。



在上图中，与黄色星形相交的线条数量为 **一个**，即红色线。但是，与黄色框相交的要素的边界框为 **两个**，即红色和蓝色框。

数据库有效回答“哪些线与黄色星形相交”的问题的方法是，首先使用索引（速度非常快）回答“哪些框与黄色框相交”的问题，然后**仅针对第一次测试返回的那些要素**进行“哪些线与黄色星形相交”的精确计算。对于大型表，这种先评估近似索引，然后进行精确测试的“两遍”系统可以大大减少回答查询所需的计算量。

PostGIS 和 Oracle Spatial 共享相同的“R 树”空间索引结构。**R 树将数据分解为矩形、子矩形和子子矩形等**。它是一种*自调整索引结构*，可自动处理可变数据密度、不同数量的对象重叠和对象大小。



6.3 自动使用空间索引函数

如果存在空间索引，只有一部分函数会自动使用它。

- ST_Intersects ST_Contains ST_Within ST_DWithin
- ST_ContainsProperly ST_CoveredBy
- ST_Covers ST_Overlaps ST_Crosses ST_DFullyWithin
- ST_3DIntersects ST_3DDWithin ST_3DDFullyWithin ST_LineCrossingDirection

- ST_OrderingEquals　ST_Equals

前四个是在查询中最常用的，并且 ST_DWithin 对于执行“在距离内”或“在半径内”的样式查询非常重要，同时仍然可以从索引中获得性能提升。

6.4 仅索引查询

PostGIS 中大多数常用函数（**ST_Contains**、**ST_Intersects**、**ST_DWithin** 等）会自动包含索引过滤器。但是，某些函数（例如，**ST_Relate**）不包含索引过滤器。为了将索引加速添加到不自动包含索引过滤器其他函数（最常见的是 ST_Relate），常通过添加必要的仅索引子句。

要使用索引进行边界框搜索（且不进行过滤），请使用 **&&** 运算符。对于几何图形，**&&** 运算符表示“边界框重叠或接触”。以下对比一下“西村”人口的仅索引查询和更精确的查询。仅索引查询使用 **&&**，

```
SELECT Sum(popn_total)    FROM nyc_neighborhoods neighborhoods
        JOIN nyc_census_blocks blocks  ON neighborhoods.geom && blocks.geom
WHERE neighborhoods.name = 'West Village';
```

⇨ *49821*

➢ **再使用更精确的 ST_Intersects 函数执行相同的查询：**

```
SELECT Sum(popn_total)  FROM nyc_neighborhoods neighborhoods
          JOIN nyc_census_blocks blocks ON ST_Intersects(neighborhoods.geom, blocks.geom)
WHERE neighborhoods.name = 'West Village';
```

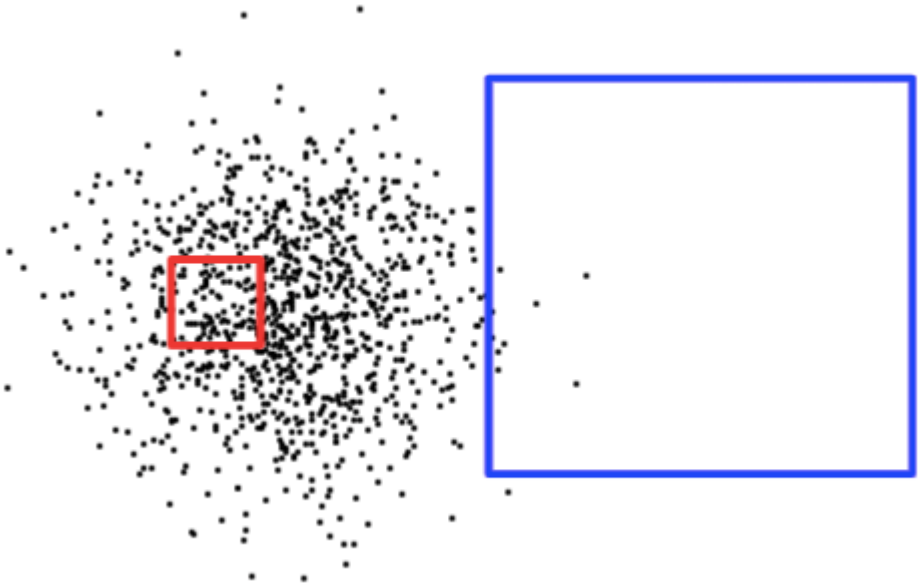
⇨ *26718*

答案少得多！第一个查询对边界框与社区边界框相交的每个区块进行了求和；第二个查询仅对与社区本身相交的那些区块进行了求和。

6.5 分析与清理

PostgreSQL 查询计划器会智能地选择何时使用或不使用索引来评估查询。与直觉相反，执行索引搜索并不总是更快：如果搜索将返回表中的每个记录，则遍历索引树以获取每个记录实际上比从头开始顺序读取整个表要慢。

我们知道，查询矩形的大小不足以确定查询将返回大量还是少量记录。在下面，红色正方形很小，但将比蓝色正方形返回更多的记录。



为了弄清楚它正在处理的情况（读取表的一小部分与读取表的大部分），PostgreSQL 会保留有关每个索引表列中数据分布的统计信息。默认情况下，PostgreSQL 会定期收集统计信息。但是，对于在短时间内大幅更改表的内容，则统计信息将不是最新的。

（1）分析命令： **ANALYZE** nyc_census_blocks;

为了确保统计信息与您的表内容匹配，明智的做法是在表中进行批量数据加载和删除后运行 ANALYZE 命令。这将强制统计系统收集所有索引列的数据。ANALYZE 命令要求 PostgreSQL 遍历表并更新其用于查询计划估计的内部统计信息。

（2）清理命令： **VACUUM ANALYZE** nyc_census_blocks;

值得注意的是，仅仅创建索引不足以使 PostgreSQL 有效地使用它。每当对表发出大量 UPDATE、INSERT 或 DELETE 时，都必须执行 VACUUM。 VACUUM 命令要求 PostgreSQL 回收因记录更新或删除而留在表页中的任何未使用空间。

清理对于数据库的有效运行至关重要，因此 PostgreSQL 默认提供“自动清理”功能。自动清理会在由活动级别确定的合理时间间隔内对表进行清理（回收空间）和分析（更新统计信息）。虽然这对于高度事务性的数据库至关重要，但在添加索引或批量加载数据后，不建议等待自动清理运行。每当执行批量更新时，都应手动运行 VACUUM。

可以根据需要手动执行清理：**VACUUM ANALYZE** nyc_census_blocks;

值得注意的是，发出 VACUUM 命令不会更新数据库统计信息；同样，发出 ANALYZE 命令不会恢复未使用的表行。这两个命令都可以针对整个数据库、单个表或单个列运行。