**SP-GIST索引介绍**

SP-GiST指空间分区(Space-Partitioned)的GiST，有助于开发四叉树，KD树，基数树(radix tree)等非平衡数据结构，它主要是通过一些新的索引算法来提高GiST索引在某种情况下的性能。它划分的搜索空间不存在重叠，因此，SP-GiST访问方法的思想是将值域拆分为不重叠的子域，每个子域又可以拆分。

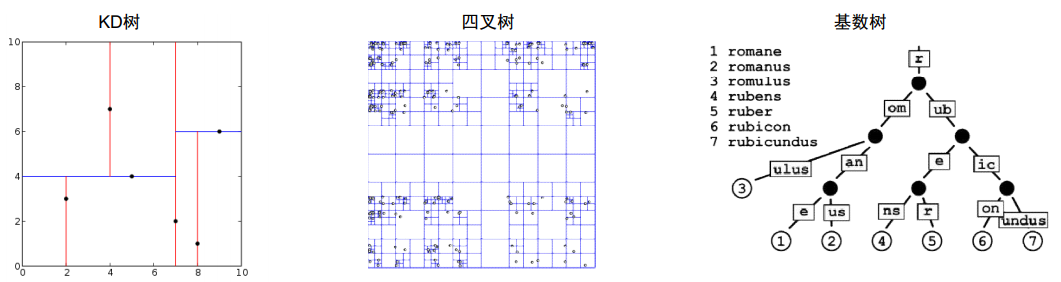
SP-GiST 和GiST都是通用的搜索树

SP-GiST的叶子节点包含索引类型的值和行（TID）的引用。此外，叶子节点可以分组为列表。因此，中间节点不仅可以引用一个值，还可以引用整个列表。

PostgreSQL内建的SP-GiST索引操作符类

|  |  |  |  |
| --- | --- | --- | --- |
| 操作符类名 | 被索引的数据类型 | 索引支持的操作符 | 索引算法 |
| kd\_point\_ops | point | << <@ <^ >> >^ ~= | KD树 |
| quad\_point\_ops | point | << <@ <^ >> >^ ~= | 四叉树 |
| range\_ops | any range type | && &< &> -|- << <@ = >> @> | 四叉树 |
| text\_ops | text | < <= = > >= ~<=~ ~<~ ~>=~ ~>~ | 基数树(radix tree) |

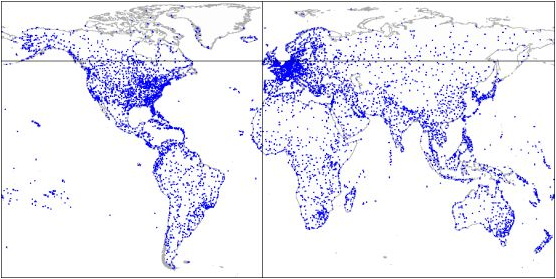
注: quad\_point\_ops是point类型缺省的SP-GiST索引操作符类



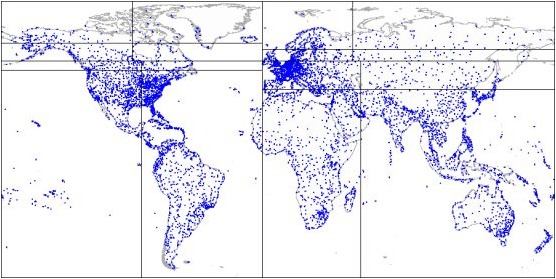
**SP-GIST应用案例1-四叉树**

四叉树用于索引平面中的点，它将区域相对于中心点递归地分为四个部分（象限）。

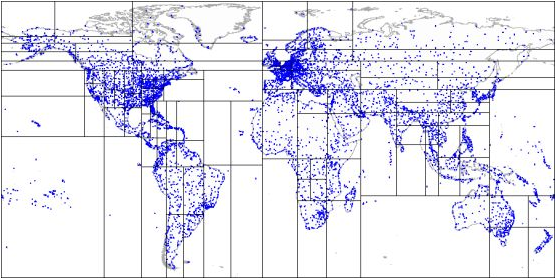
首先，将平面划分为四个象限...



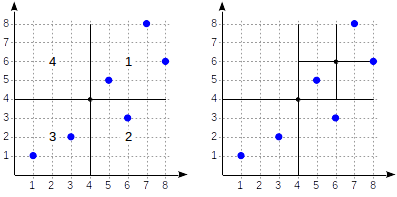
然后，划分每个象限...，



依此类推，直到获得最终分区。

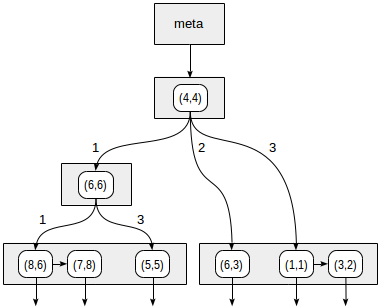


看看分区是什么样的：



象限的编号如图所示。从左到右按照完全相同的顺序放置叶子节点。每个中间节点最多引用四个叶子节点。

每个参考都可以用象限编号标记。



示例：

create table points(p point);

insert into points(p) values (point '(1,1)'), (point '(3,2)'), (point'(6,3)'),(point '(5,5)'), (point '(7,8)'), (point '(8,6)');

create index on points using spgist(p);

默认情况下使用 quad\_point\_ops运算符类，包含以下运算符：

select amop.amopopr::regoperator, amop.amopstrategy

from pg\_opclass opc, pg\_opfamily opf, pg\_am am, pg\_amop amop

where opc.opcname = 'quad\_point\_ops'

and opf.oid = opc.opcfamily

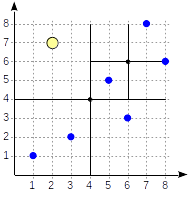
and am.oid = opf.opfmethod

and amop.amopfamily = opc.opcfamily

and am.amname = 'spgist'

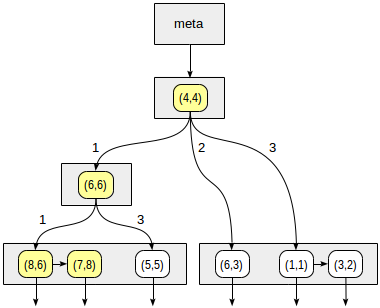
and amop.amoplefttype = opc.opcintype;

select \* from points where p >^ point '(2,7)';  --如何执行（找到位于给定点上方的所有点）。



对于运算符>^，从根节点开始，将点（2,7）与节点（4,4）的中心点进行比较，并选择可能包含所查找点的象限（第一和第四象限）。

在与第一象限相对应的节点中，中心点是（6,6），我们需要再次浏览第一和第四象限。



叶子节点（8,6）和（7,8）的列表对应于第一象限，其中只有点（7,8）满足查询条件。对第四象限的引用为空。在中间节点（4,4）中，对第四象限的引用也为空，这样就完成了搜索。

set enable\_seqscan = off;

explain select \* from points where p >^ point '(2,7)';

**SP-GIST使用案例2-：k维树**

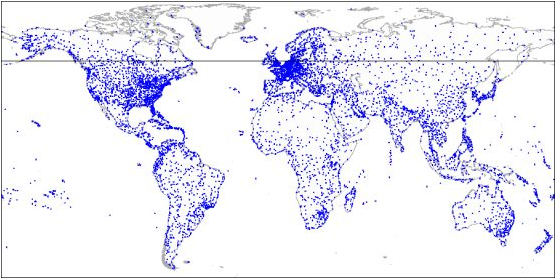
对于平面中的相同点，可以使用另一种划分空间的方法。通过被索引的第一个点画一条水平线，将平面分为两部分：上部和下部。

将要索引的第二个点画一条垂直线，将这部分分为两部分：右和左。再次通过下一个点绘制一条水平线，并通过下一个点绘制一条垂直线，依此类推。

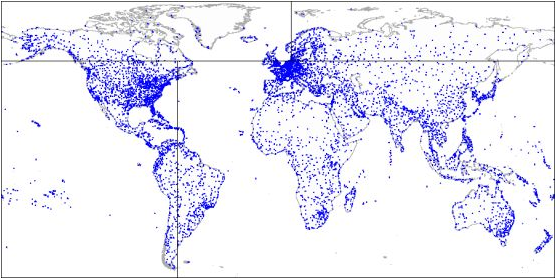
以这种方式构建的树的所有中间节点将只有两个叶子节点。这两个引用中的每一个都可以指向层次结构中下一个中间节点或叶节点列表。这种方法可以很容易地推广到k维空间，因此，这些树也称为k维树（k-D树）。

通过机场示例来解释该方法：

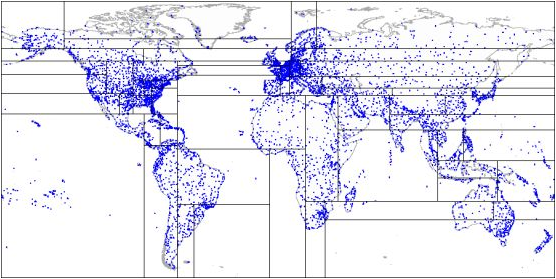
首先把机场分成上下两部分。



然后将每一部分分为左右两部分



依此类推，直到获得最终分区。



要像这样使用分区，需要在创建索引时显式地指定运算符类 kd\_point\_ops，它包括与 default 类和 quad\_point\_ops 完全相同的运算符。
create index on points using spgist(p kd\_point\_ops);

select \* from points where p >^ point '(2,7)';

**SP-GIST使用案例3-基数树**

还可以使用SP-GiST实现字符串的基数树。基数树的想法是要索引的字符串没有完全存储在叶节点中，而是通过将存储在该节点中的值连接到根节点来获得的。

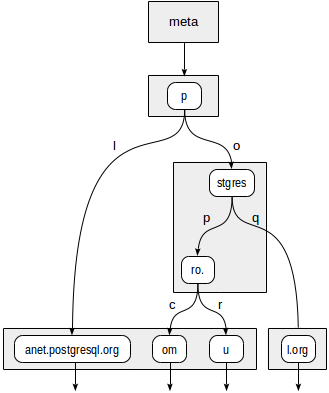
示例：索引站点URL：postgrespro.ru，postgrespro.com，postgresql.org 和 planet.postgresql.org。

create table sites(url text);

insert into sites values('postgrespro.ru'),('postgrespro.com'),('postgresql.org'),('planet.postgresql.org');

create index on sites using spgist(url);

这棵树看起来是这样的:



树的中间节点存储所有叶子节点共有的前缀。例如，在 stgres 的叶子节点中，值以p + o + stgres 开头。

与四叉树不同，指向叶子节点的每个指针都另外用一个字符标记（更确切地说，用两个字节标记）。text\_ops 运算符类支持类似B树的运算符：=、> 和 <：

select amop.amopopr::regoperator, amop.amopstrategy

from pg\_opclass opc, pg\_opfamily opf, pg\_am am, pg\_amop amop

where opc.opcname = 'text\_ops'

and opf.oid = opc.opcfamily

and am.oid = opf.opfmethod

and amop.amopfamily = opc.opcfamily

and am.amname = 'spgist'

and amop.amoplefttype = opc.opcintype;

带波浪号的运算符的区别在于，它们操作字节而不是字符。

以基数树的形式表示的结果可能会比B树要紧凑得多，因为这些值并未完全存储，而是根据需要在遍历树的过程中重新构建。

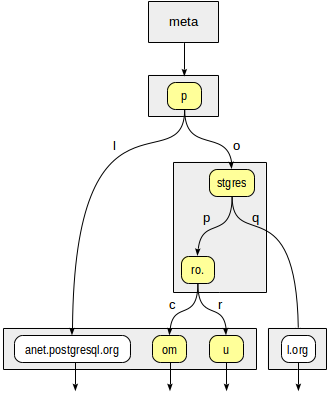
select \* from sites where url like ‘postgresp%ru’;

explain select \* from sites where url like'postgresp%ru';

实际上，索引用于查找大于或等于 postgresp 但小于 postgresq 的值，然后从结果中选择匹配的值。首先，必须确定需要下降到 p 根的哪个叶子节点。

有两个选项：p+ l（无需下降）和p + o + stgres（继续下降）。对于 stgres 节点，需要再次检查postgres + p + ro.（继续下降）和postgres+ q（不需要下降）。

对于 ro.节点及其所有子叶节点都返回 yes，因此将返回两个值：postgrespro.com 和 postgrespro.ru。



最好选出以ru结尾的值：postgrespro.ru

**NULL值**

SP-GIST支持NULL：

explain select \* from sites where url is null;