阿里云
云数据库 MySQL 版

时空数据库

文档版本：20190905

法律声明

阿里云提醒您在阅读或使用本文档之前仔细阅读、充分理解本法律声明各条款的内容。如果您阅读或使用本文档，您的阅读或使用行为将被视为对本声明全部内容的认可。

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。
2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。
3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。
4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的”现状“、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。
5. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含”阿里云”、Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。
6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

格式	说明	样例
	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	 **禁止:** 重置操作将丢失用户配置数据。
	该类警示信息可能导致系统重大变更甚至故障，或者导致人身伤害等结果。	 **警告:** 重启操作将导致业务中断，恢复业务所需时间约10分钟。
	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	 **说明:** 您也可以通过按Ctrl + A选中全部文件。
>	多级菜单递进。	设置 > 网络 > 设置网络类型
粗体	表示按键、菜单、页面名称等UI元素。	单击 确定。
`courier` 字体	命令。	执行 `cd /d C:/windows` 命令，进入Windows系统文件夹。
*## *	表示参数、变量。	`bae log list --instanceid` *Instance_ID*
[]或者[a|b]	表示可选项，至多选择一个。	`ipconfig` *[-all|-t]*
{}或者{a|b}	表示必选项，至多选择一个。	`swich` *{stand | slave}*

目录

法律声明.....I
通用约定.....I
1 简介..... 1
2 模型..... 2
2.1 几何模型.....2
2.2 栅格模型.....6
2.3 路径模型.....8
2.4 点云模型.....10
2.5 轨迹模型.....13
3 使用进阶.....17
3.1 开启时空并行查询.....17
3.2 开启GPU加速计算..... 18
4 Raster SQL参考.....21
4.1 基本概念.....21
4.2 Raster创建.....21
4.2.1 ST_CreateRast..... 22
4.3 导入导出.....22
4.3.1 ST_ImportFrom.....23
4.3.2 ST_ExportTo..... 23
4.4 金字塔操作.....24
4.4.1 ST_BuildPyramid..... 24
4.4.2 ST_deletePyramid..... 25
4.4.3 ST_BestPyramidLevel.....25
4.5 坐标系统转换.....26
4.5.1 ST_Rast2WorldCoord.....26
4.5.2 ST_World2RastCoord.....27
4.6 像素值操作.....27
4.6.1 ST_ClipDimension.....27
4.6.2 ST_Clip..... 28
4.6.3 ST_ClipToRast..... 30
4.6.4 ST_Values.....32
4.6.5 ST_Update.....34
4.6.6 ST_MosaicFrom.....34
4.6.7 ST_MosaicTo..... 35
4.7 Overview操作.....35
4.7.1 ST_BuildOverview.....36
4.7.2 ST_UpdateOverview.....36
4.7.3 ST_EraseOverview.....37
4.8 DEM操作.....37

4.8.1 ST_Aspect..... 38
4.8.2 ST_Slope..... 39
4.8.3 ST_Hillshade..... 40
4.8.4 ST_Overflow..... 41
4.8.5 ST_Flow_direction..... 42
4.9 属性的查询与更新..... 42
4.9.1 ST_Name..... 43
4.9.2 ST_SetName..... 43
4.9.3 ST_MetaData..... 44
4.9.4 ST_Width..... 44
4.9.5 ST_Height..... 45
4.9.6 ST_NumBands..... 45
4.9.7 ST_Value..... 46
4.9.8 ST_RasterID..... 46
4.9.9 ST_CellDepth..... 47
4.9.10 ST_CellType..... 47
4.9.11 ST_InterleavingType..... 48
4.9.12 ST_TopPyramidLevel..... 48
4.9.13 ST_Extent..... 49
4.9.14 ST_Srid..... 49
4.9.15 ST_SetSrid..... 50
4.9.16 ST_Georeference..... 50
4.9.17 ST_IsGeoreferenced..... 51
4.9.18 ST_UnGeoreference..... 51
4.9.19 ST_SetGeoreference..... 52
4.9.20 ST_NoData..... 53
4.9.21 ST_SetNoData..... 53
4.9.22 ST_ColorTable..... 54
4.9.23 ST_SetColorTable..... 55
4.9.24 ST_Statistics..... 55
4.9.25 ST_SetStatistics..... 56
4.9.26 ST_SummaryStats..... 56
4.9.27 ST_ColorInterp..... 57
4.9.28 ST_SetColorInterp..... 58
4.9.29 ST_Histogram..... 59
4.9.30 ST_SetHistogram..... 60
4.9.31 ST_BuildHistogram..... 62
4.10 辅助函数..... 63
4.10.1 ST_CheckGPU..... 63
5 PointCloud SQL参考..... 64
5.1 构造函数..... 64
5.1.1 ST_makePoint..... 64
5.1.2 ST_makePatch..... 64
5.1.3 ST_Patch..... 65
5.2 属性函数..... 65

5.2.1 ST_asText....65
5.2.2 ST_pcID....66
5.2.3 ST_get....66
5.2.4 ST_numPoints....67
5.2.5 ST_summary....67
5.3 对象操作....67
5.3.1 ST_compress....68
5.3.2 ST_unCompress....69
5.3.3 ST_union....69
5.3.4 ST_explode....70
5.3.5 ST_patchAvg....70
5.3.6 ST_patchMax....71
5.3.7 ST_patchMin....71
5.3.8 ST_patchAvg....72
5.3.9 ST_patchMax....72
5.3.10 ST_patchMin....73
5.3.11 ST_filterGreaterThan....73
5.3.12 ST_filterLessThan....74
5.3.13 ST_filterEquals....74
5.3.14 ST_filterBetween....75
5.3.15 ST_pointN....75
5.3.16 ST_isSorted....76
5.3.17 ST_sort....76
5.3.18 ST_range....77
5.3.19 ST_setPcid....77
5.3.20 ST_transform....78
5.3.21 ST_envelopeGeometry....78
5.3.22 ST_boundingDiagonalGeometry....79
5.4 OGC WKB操作....79
5.4.1 ST_asBinary....80
5.4.2 ST_envelopeAsBinary....80
5.4.3 ST_boundingDiagonalAsBinary....81
5.5 空间关系判断....81
5.5.1 ST_intersects....81
5.6 空间处理....82
5.6.1 ST_intersection....82
6 Trajectory SQL参考....83
6.1 基本概念....83
6.2 构造函数....84
6.2.1 构造函数概述....84
6.2.2 ST_makeTrajectory....84
6.2.3 ST_append....89
6.3 编辑与处理函数....90
6.3.1 ST_Compress....90
6.3.2 ST_CompressSED....94

6.3.3 ST_attrDeduplicate....95
6.3.4 ST_sort.... 96
6.3.5 ST_deviation.... 97
6.4 属性元数据....97
6.4.1 ST_attrDefinition....97
6.4.2 ST_attrSize....98
6.4.3 ST_attrName.... 98
6.4.4 ST_attrType....99
6.4.5 ST_attrLength....101
6.4.6 ST_attrNullable.... 102
6.5 事件函数.... 103
6.5.1 ST_addEvent....103
6.5.2 ST_eventTimes.... 104
6.5.3 ST_eventTime....104
6.5.4 ST_eventTypes....105
6.5.5 ST_eventType.... 106
6.6 属性函数.... 106
6.6.1 ST_startTime....106
6.6.2 ST_endTime.... 107
6.6.3 ST_trajectorySpatial.... 107
6.6.4 ST_trajectoryTemporal.... 107
6.6.5 ST_trajAttrs.... 108
6.6.6 ST_attrIntMax.... 108
6.6.7 ST_attrIntMin....109
6.6.8 ST_attrIntAverage....110
6.6.9 ST_attrFloatMax.... 110
6.6.10 ST_attrFloatMin....111
6.6.11 ST_attrFloatAverage....112
6.6.12 ST_leafType....112
6.6.13 ST_leafCount.... 113
6.6.14 ST_duration....113
6.6.15 ST_Distance（下线）....114
6.6.16 ST_timeAtPoint....114
6.6.17 ST_pointAtTime....114
6.6.18 ST_velocityAtTime....115
6.6.19 ST_accelerationAtTime.... 115
6.6.20 ST_bearingAtTime（下线）.... 116
6.6.21 ST_accuraryAtTime（下线）....116
6.6.22 ST_timeToDistance....117
6.6.23 ST_timeAtDistance....117
6.6.24 ST_cumulativeDistanceAtTime.... 117
6.6.25 ST_timeAtCumulativeDistance....118
6.6.26 ST_subTrajectory....118
6.6.27 ST_subTrajectorySpatial.... 119
6.6.28 ST_samplingInterval....120

6.6.29 ST_trajAttrsAsText..... 120
6.6.30 ST_trajAttrsAsInteger..... 122
6.6.31 ST_trajAttrsAsDouble..... 122
6.6.32 ST_trajAttrsAsBool..... 123
6.6.33 ST_trajAttrsAsTimestamp..... 124
6.6.34 ST_attrIntFilter..... 124
6.6.35 ST_attrFloatFilter..... 126
6.6.36 ST_attrTimestampFilter..... 127
6.6.37 ST_attrNullFilter..... 128
6.6.38 ST_attrNotNullFilter..... 129
6.6.39 ST_trajAttrsMeanMax..... 130
6.7 空间关系判断..... 132
6.7.1 ST_intersects..... 132
6.7.2 ST_equals..... 132
6.7.3 ST_distanceWithin..... 133
6.8 空间处理..... 133
6.8.1 ST_intersection..... 134
6.8.2 ST_difference..... 134
6.9 空间统计..... 135
6.9.1 ST_nearestApproachPoint..... 135
6.9.2 ST_nearestApproachDistance..... 135
6.10 时空关系判断..... 136
6.10.1 ST_intersects..... 136
6.10.2 ST_equals..... 137
6.10.3 ST_distanceWithin..... 137
6.10.4 ST_durationWithin..... 138
6.11 时空处理..... 138
6.11.1 ST_intersection..... 139
6.12 时空统计..... 139
6.12.1 ST_nearestApproachPoint..... 139
6.12.2 ST_nearestApproachDistance..... 140
6.13 距离测量..... 140
6.13.1 ST_length..... 141
6.13.2 ST_euclideanDistance..... 141
6.13.3 ST_mdistance..... 142
6.14 相似度分析..... 142
6.14.1 ST_lcsSimilarity..... 142
6.14.2 ST_lcsDistance..... 144
6.14.3 ST_lcsSubDistance..... 146
6.15 变量..... 148
6.15.1 ganos.trajectory.attr_string_length..... 148
7 Trajectory 最佳实践..... 149
8 Trajecotry 常见问题..... 151
9 服务发布..... 155

9.1 GeoServer..... 155
9.1.1 GeoServer简介..... 155
9.1.2 发布几何数据..... 155
9.1.3 发布栅格数据..... 159
10 桌面应用..... 163
10.1 QGIS访问Ganos..... 163
10.2 uDig访问Ganos..... 165
10.3 OpenJump访问Ganos..... 168

1 简介

数据（Spatial/Spatio-temproal Data，以下统称时空数据)是带有时间/空间位置信息的图形图像数据，用来表示事物的位置、形态、变化及大小分布等多维信息。

概述

空间/时空ApsaraDB PostgreSQL Ganos时空引擎（以下简称Ganos）提供一系列的数据类型、函数和存储过程，用于在阿里云关系型数据库（Relational Database Service，简称 RDS）中对空间/时空数据进行高效的存储、索引、查询和分析计算。

本文档向您介绍如何使用Ganos对时空数据进行管理和分析。

如果您需要获取人工帮助，可以拨打技术支持电话95187或者在RDS管理控制台的右上角选择工单 > 提交工单。如果业务复杂，您也可以购买支持计划，获取由IM企业群、技术服务经理（TAM）、服务经理等提供的专属支持。

定价

目前云数据库时空引擎Ganos包含在RDS For PostgreSQL版本中，免费进行使用。

2 模型

2.1 几何模型

Ganos Geometry是对象关系型数据库PostgreSQL的一个空间几何扩展，Ganos Geometry遵循OpenGIS规范，使PostgreSQL增加了存储和管理2D（X，Y）、3D（X，Y，Z）、4D（X，Y，Z，M）空间几何数据的能力，并提供了空间几何对象、索引、函数和操作符等丰富功能。

概述

几何模型完全兼容PostGIS接口，支持已有应用的平滑迁移。

快速入门

- 创建扩展

```
--创建几何扩展
Create extension ganos_geometry cascade;

--创建几何拓扑扩展
Create extension ganos_geometry_topology;

--创建sfcgal插件扩展
Create extension ganos_geometry_sfcgal;
```

- 创建几何表

```
--方式一：直接创建带geometry字段的表
CREATE TABLE ROADS ( ID int4, ROAD_NAME varchar(25), geom geometry(
LINESTRING,4326) );

--方式二：先创建普通表，再附加几何字段
CREATE TABLE ROADS ( ID int4, ROAD_NAME varchar(25) );
SELECT AddGeometryColumn( 'roads', 'geom', 4326, 'LINESTRING', 2);
```

- 添加几何约束

```
ALTER TABLE ROADS ADD CONSTRAINT geometry_valid_check CHECK (
ST_IsValid(geom));
```

- 导入几何数据

```
INSERT INTO roads (id, geom, road_name)
  VALUES (1,ST_GeomFromText('LINESTRING(191232 243118,191108 243242
)',-1),'北五环');
INSERT INTO roads (id, geom, road_name)
  VALUES (2,ST_GeomFromText('LINESTRING(189141 244158,189265 244817
)',-1),'东五环');
INSERT INTO roads (id, geom, road_name)
  VALUES (3,ST_GeomFromText('LINESTRING(192783 228138,192612 229814
)',-1),'南五环');
```

```
INSERT INTO roads (id, geom, road_name)
  VALUES (4,ST_GeomFromText('LINESTRING(189412 252431,189631 259122
)',-1),'西五环');
INSERT INTO roads (id, geom, road_name)
  VALUES (5,ST_GeomFromText('LINESTRING(190131 224148,190871 228134
)',-1),'东长安街');
INSERT INTO roads (id, geom, road_name)
  VALUES (6,ST_GeomFromText('LINESTRING(198231 263418,198213 268322
)',-1),'西长安街');
```

- 查询几何对象信息

```
SELECT id, ST_AsText(geom) AS geom, road_name FROM roads;

-------------------------------
     id | geom                                        | road_name
--------+---------------------------------------------+-----------
      1 | LINESTRING(191232 243118,191108 243242) | 北五环
      2 | LINESTRING(189141 244158,189265 244817) | 东五环
      3 | LINESTRING(192783 228138,192612 229814) | 南五环
      4 | LINESTRING(189412 252431,189631 259122) | 西五环
      5 | LINESTRING(190131 224148,190871 228134) | 东长安街
      6 | LINESTRING(198231 263418,198213 268322) | 西长安街
(6 rows)
```

- 创建索引

```
--GiST索引
CREATE INDEX [indexname] ON [tablename] USING GIST ( [geometryfield
] );
CREATE INDEX [indexname] ON [tablename] USING GIST ([geometryfield]
gist_geometry_ops_nd);
VACUUM ANALYZE [table_name] [(column_name)];

--举例
Create INDEX sp_geom_index ON ROADS USING GIST(geom);
VACUUM ANALYZE ROADS (geom);

--创建BRIN索引
CREATE INDEX [indexname] ON [tablename] USING BRIN ( [geometryfield
] );
CREATE INDEX [indexname] ON [tablename] USING BRIN ([geometryfield]
brin_geometry_inclusion_ops_3d);
CREATE INDEX [indexname] ON [tablename] USING BRIN ([geometryfield]
brin_geometry_inclusion_ops_4d);
--创建指定大小的brin索引
CREATE INDEX [indexname] ON [tablename] USING BRIN ( [geometryfield
] ) WITH (pages_per_range = [number]);
```

- 几何对象存取

```
---判断空间几何对象是否是简单要素类型
 SELECT ST_IsSimple(ST_GeomFromText('POLYGON((1 2, 3 4, 5 6, 1 2
))'));
 st_issimple
-------------
 t
(1 row)

 SELECT ST_IsSimple(ST_GeomFromText('LINESTRING(1 1,2 2,2 3.5,1 3,1
2,2 1)'));
 st_issimple
```

```
-------------
 f
(1 row)


--查询地形中拥有环岛且面积最大的城市
SELECT gid, name, ST_Area(the_geom) AS area
FROM bc_municipality
WHERE ST_NRings(the_geom) > 1
ORDER BY area DESC LIMIT 1;

gid  | name         | area
-----+--------------+------------------
12   | 安宁市        | 257374619.430216
(1 row)
```

- 空间测量、空间分析和空间关系判断

```
--创建表bc_roads
Create table bc_roads (gid serial, name varchar, the_geom geometry);

--创建表bc_municipality
Create table bc_municipality(gid serial, code integer, name varchar
, the_geom geometry);

--长度计算
SELECT sum(ST_Length(the_geom))/1000 AS km_roads FROM bc_roads;

km_roads
------------------
70842.1243039643
(1 row)

--面积计算
SELECT ST_Area(the_geom)/10000 AS hectares FROM bc_municipality
WHERE name = 'PRINCE GEORGE';

hectares
------------------
32657.9103824927
(1 row)
--使用ST_Contains函数
SELECT  m.name, sum(ST_Length(r.the_geom))/1000 as roads_km
FROM
  bc_roads AS r, bc_municipality AS m
WHERE
  ST_Contains(m.the_geom,r.the_geom)
GROUP BY m.name
ORDER BY roads_km;

name                         | roads_km
-----------------------------+------------------
SURREY                       | 1539.47553551242
VANCOUVER                    | 1450.33093486576
LANGLEY DISTRICT             | 833.793392535662
BURNABY                      | 773.769091404338
PRINCE GEORGE                | 694.37554369147


--使用ST_Covers函数
SELECT ST_Covers(smallc,smallc) As smallinsmall,
    ST_Covers(smallc, bigc) As smallcoversbig,
    ST_Covers(bigc, ST_ExteriorRing(bigc)) As bigcoversexterior,
```

```
    ST_Contains(bigc, ST_ExteriorRing(bigc)) As bigcontainsexterior
FROM (SELECT ST_Buffer(ST_GeomFromText('POINT(1 2)'), 10) As smallc,
    ST_Buffer(ST_GeomFromText('POINT(1 2)'), 20) As bigc) As foo;
    --Result
 smallinsmall | smallcoversbig | bigcoversexterior | bigcontain
sexterior
--------------+----------------+-------------------
+--------------------
 t             | f              | t                 | f
(1 row)

--使用ST_Disjoint函数
SELECT ST_Disjoint('POINT(0 0)'::geometry, 'LINESTRING ( 2 0, 0 2
 )'::geometry);
 st_disjoint
---------------
 t
(1 row)
SELECT ST_Disjoint('POINT(0 0)'::geometry, 'LINESTRING ( 0 0, 0 2
 )'::geometry);
 st_disjoint
---------------
 f
(1 row)

--使用ST_Overlaps函数
SELECT ST_Overlaps(a,b) As a_overlap_b,
    ST_Crosses(a,b) As a_crosses_b,
        ST_Intersects(a, b) As a_intersects_b, ST_Contains(b,a) As
b_contains_a
FROM (SELECT ST_GeomFromText('POINT(1 0.5)') As a, ST_GeomFromText('
LINESTRING(1 0, 1 1, 3 5)')  As b)
    As foo

a_overlap_b | a_crosses_b | a_intersects_b | b_contains_a
------------+-------------+----------------+--------------
f           | f           | t              | t

--使用ST_Relate函数
SELECT ST_Relate(ST_GeometryFromText('POINT(1 2)'), ST_Buffer(
ST_GeometryFromText('POINT(1 2)'),2), '0FFFFF212');
st_relate
-----------
t

--使用ST_Touches函数
SELECT ST_Touches('LINESTRING(0 0, 1 1, 0 2)'::geometry, 'POINT(1 1
)'::geometry);
 st_touches
------------
 f
(1 row)

SELECT ST_Touches('LINESTRING(0 0, 1 1, 0 2)'::geometry, 'POINT(0 2
)'::geometry);
 st_touches
------------
 t
(1 row)

--使用ST_Within函数
SELECT ST_Within(smallc,smallc) As smallinsmall,
    ST_Within(smallc, bigc) As smallinbig,
    ST_Within(bigc,smallc) As biginsmall,
```

```
    ST_Within(ST_Union(smallc, bigc), bigc) as unioninbig,
    ST_Within(bigc, ST_Union(smallc, bigc)) as biginunion,
    ST_Equals(bigc, ST_Union(smallc, bigc)) as bigisunion
FROM
(
SELECT ST_Buffer(ST_GeomFromText('POINT(50 50)'), 20) As smallc,
    ST_Buffer(ST_GeomFromText('POINT(50 50)'), 40) As bigc) As foo;
--Result
 smallinsmall | smallinbig | biginsmall | unioninbig | biginunion |
bigisunion
--------------+------------+------------+------------+------------
+------------
 t            | t          | f          | t          | t          |
t
(1 row)
```

- 删除扩展

```
Drop extension ganos_geometry cascade;
```

SQL参考

详细SQL手册请参考 PostGIS官方手册。

2.2 栅格模型

栅格数据由按行和列组织的像元（也称为像素）矩阵组成，其中每个像元都包含一个信息值（例如温度）。

概述

栅格数据可以是数字航空像片、卫星影像、数字图片或甚至扫描的地图。

Ganos Raster通过在RDS for PostgreSQL中实现栅格数据模型，可以借助于数据库的技术和方法高效地对栅格数据进行存储和分析操作。

快速入门

- 创建扩展

```
Create Extension Ganos_Raster cascade;
```

- 创建栅格表

```
Create Table raster_table(id integer, raster_obj raster);
```

- 从OSS中导入栅格数据

```
Insert into raster_table Values(1, ST_ImportFrom('chunk_table','OSS
://ABCDEFG:1234567890@oss-cn.aliyuncs.com/mybucket/data/4.tif'))
```

- 查询栅格对象信息

```
Select ST_Height(raster_obj),ST_Width(raster_obj) From raster_table
Where id = 1;
```

- 创建金字塔

```
Update raster_table Set raster_obj = ST_BuildPyramid(raster_obj)
Where id = 1;
```

- 根据视口的世界坐标范围，长和宽来计算最佳的金字塔层级

```
Select ST_BestPyramidLevel(raster_obj, '((128.0, 30.0),(128.5, 30.5
))', 800, 600) from raster_table where id = 10;

----------------------
3
```

- 获取栅格指定范围的像素矩阵

```
Select ST_Clip(raster_obj, 0, '((128.980,30.0),(129.0,30.2))', '
World') From raster_table Where id = 1;
```

- 计算裁剪区域的像素坐标范围

```
Select ST_ClipDimension(raster_obj, 2, '((128.0, 30.0),(128.5, 30.5
))') from rater_table where id = 10;

-------------------------------------
'((600, 720),(200, 300))'
```

- GPU加速

  当用户购买包含GPU的RDS实例时，自动开启GPU并行加速计算，用户无需设置额外的参数。

  以创建金字塔为例，介绍支持GPU加速计算的接口ST_BuildPyramid：

```

--创建影像金字塔过程中重采样支持GPU加速，重采样类型可以为Near、Average、Cubic
、Bilinear中的任意一种
Update raster_table set rast = ST_BuildPyramid(rast) where id = 1;
Update raster_table set rast = ST_BuildPyramid(rast, 6, 'Bilinear')
where id = 1;
```

```
Update raster_table set rast = ST_BuildPyramid(rast, 6, 'Bilinear',
 'chunk_table') where id = 1;
```

关于GPU的更多介绍可参考#unique_7章节。

- 删除扩展

```
Drop Extension Ganos_raster cascade;
```

SQL参考

详细SQL手册请参见Raster SQL 参考 。

2.3 路径模型

路径数据是由Edge和Node构成的几何网络图，主要用于构建道路和交通网络。

概述

Ganos Networking是对象关系型数据库PostgreSQL的一个扩展，提供了一系列的函数和存储过程，用于根据代价模型查找最快、最短甚至是最优的路径。如果代价是时间，则最佳路线为最快路线。如果代价是距离，则最佳路径为最短路径。路径模型用于解决基于道路网的路径规划问题、电子地图GPS导航路径搜索规划问题、路由问题等。路径模型完全兼容PGRouting接口，支持已有应用的平滑迁移。

快速入门

- 创建扩展

```
Create Extension Ganos_Networking cascade;
```

- 创建表

```
CREATE TABLE edge_table (
    id BIGSERIAL,
    dir character varying,
    source BIGINT,
    target BIGINT,
    cost FLOAT,
    reverse_cost FLOAT,
    capacity BIGINT,
    reverse_capacity BIGINT,
    category_id INTEGER,
    reverse_category_id INTEGER,
    x1 FLOAT,
    y1 FLOAT,
    x2 FLOAT,
    y2 FLOAT,
    the_geom geometry
```

```
);
```

- 插入记录

```
INSERT INTO edge_table (
    category_id, reverse_category_id,
    cost, reverse_cost,
    capacity, reverse_capacity,
    x1, y1,
    x2, y2) VALUES
(3, 1,    1,  1,  80, 130,   2,   0,    2, 1),
(3, 2,   -1,  1,  -1, 100,   2,   1,    3, 1),
(2, 1,   -1,  1,  -1, 130,   3,   1,    4, 1),
(2, 4,    1,  1, 100,  50,   2,   1,    2, 2),
(1, 4,    1, -1, 130,  -1,   3,   1,    3, 2),
(4, 2,    1,  1,  50, 100,   0,   2,    1, 2),
(4, 1,    1,  1,  50, 130,   1,   2,    2, 2),
(2, 1,    1,  1, 100, 130,   2,   2,    3, 2),
(1, 3,    1,  1, 130,  80,   3,   2,    4, 2),
(1, 4,    1,  1, 130,  50,   2,   2,    2, 3),
(1, 2,    1, -1, 130,  -1,   3,   2,    3, 3),
(2, 3,    1, -1, 100,  -1,   2,   3,    3, 3),
(2, 4,    1, -1, 100,  -1,   3,   3,    4, 3),
(3, 1,    1,  1,  80, 130,   2,   3,    2, 4),
(3, 4,    1,  1,  80,  50,   4,   2,    4, 3),
(3, 3,    1,  1,  80,  80,   4,   1,    4, 2),
(1, 2,    1,  1, 130, 100,   0.5, 3.5,  1.999999999999,3.5),
(4, 1,    1,  1,  50, 130,   3.5, 2.3,  3.5,4);
```

- 更新表属性

```
UPDATE edge_table SET the_geom = st_makeline(st_point(x1,y1),
st_point(x2,y2)),
dir = CASE WHEN (cost>0 AND reverse_cost>0) THEN 'B'
           WHEN (cost>0 AND reverse_cost<0) THEN 'FT'
           WHEN (cost<0 AND reverse_cost>0) THEN 'TF'
           ELSE '' END;
```

- 创建拓扑

```
SELECT pgr_createTopology('edge_table',0.001);
```

- 查询最短路径

```
-- 使用dijkstra算法查询最短路径
SELECT * FROM pgr_dijkstra(
    'SELECT id, source, target, cost, reverse_cost FROM edge_table',
    2, 3
);

 seq | path_seq | node | edge | cost | agg_cost
-----+----------+------+------+------+----------
   1 |        1 |    2 |    4 |    1 |        0
   2 |        2 |    5 |    8 |    1 |        1
   3 |        3 |    6 |    9 |    1 |        2
   4 |        4 |    9 |   16 |    1 |        3
   5 |        5 |    4 |    3 |    1 |        4
   6 |        6 |    3 |   -1 |    0 |        5
(6 rows)

-- 使用astar算法查询最短路径
SELECT * FROM pgr_astar(
```

```
    'SELECT id, source, target, cost, reverse_cost, x1, y1, x2, y2
FROM edge_table',
    2, 12,
    directed := false, heuristic := 2);

 seq | path_seq | node | edge | cost | agg_cost
-----+----------+------+------+------+----------
   1 |        1 |    2 |    2 |    1 |        0
   2 |        2 |    3 |    3 |    1 |        1
   3 |        3 |    4 |   16 |    1 |        2
   4 |        4 |    9 |   15 |    1 |        3
   5 |        5 |   12 |   -1 |    0 |        4
(5 rows)

-- 使用trsp 路径算法
SELECT * FROM pgr_trsp(
        'SELECT id::INTEGER, source::INTEGER, target::INTEGER, cost
FROM edge_table',
        2, 7, false, false,
        'SELECT to_cost, target_id::int4,
        from_edge || coalesce('','' || via_path, '''') AS via_path
        FROM restrictions'
    );

 seq | id1 | id2 | cost
-----+-----+-----+------
   0 |   2 |   4 |    1
   1 |   5 |  10 |    1
   2 |  10 |  12 |    1
   3 |  11 |  11 |    1
   4 |   6 |   8 |    1
   5 |   5 |   7 |    1
   6 |   8 |   6 |    1
   7 |   7 |  -1 |    0
(8 rows)
```

- 删除扩展

```
Drop Extension Ganos_Networking cascade;
```

SQL参考

详细SQL手册请参考 pgrouting 官方文档 。

2.4 点云模型

点云数据通常是由3D扫描仪扫描资料并以点的形式输出的记录，每一个点包含有三维坐标，有些可能含有颜色信息（RGB）或反射强度信息（Intensity），点云数据含有空间坐标信息，且具有数量众多、属性维度复杂的特点。

概述

Ganos PointCloud是对象关系型数据库PostgreSQL的一个扩展，使PostgreSQL能够有效快速存储和管理点云数据，并提供点云数据压缩、解压缩、属性统计等功能，同时联合Ganos Geometry模块提供点云空间分析的能力。

点云数据类型

点云数据类型主要分为两种，一种是pcpoint数据类型，一个点一行记录存储。点的维度信息在元数据中定义。另一种是pcpatch数据类型，该类型将点以集合的方式进行存储，支持压缩，减少存储空间，支持空间检索。压缩方式由元数据中的“compression”决定。

点云元数据

表pointcloud_formats记录了点云的schema（元数据）信息。元数据包括点云的属性维度，以及每个维度的数据大小、类型、名称以及解释说明等。

快速入门

- 创建扩展

```
Create extension ganos_pointcloud cascade;
Create extension ganos_pointcloud_geometry cascade;
```

- 插入点云schema

```
INSERT INTO pointcloud_formats (pcid, srid, schema) VALUES (1, 4326,
'<?xml version="1.0" encoding="UTF-8"?>
<pc:PointCloudSchema xmlns:pc="http://pointcloud.org/schemas/PC/1.1"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <pc:dimension>
    <pc:position>1</pc:position>
    <pc:size>4</pc:size>
    <pc:description>X coordinate as a long integer. You must use the
                    scale and offset information of the header to
                    determine the double value.</pc:description>
    <pc:name>X</pc:name>
    <pc:interpretation>int32_t</pc:interpretation>
    <pc:scale>0.01</pc:scale>
  </pc:dimension>
  <pc:dimension>
    <pc:position>2</pc:position>
    <pc:size>4</pc:size>
    <pc:description>Y coordinate as a long integer. You must use the
                    scale and offset information of the header to
                    determine the double value.</pc:description>
    <pc:name>Y</pc:name>
    <pc:interpretation>int32_t</pc:interpretation>
    <pc:scale>0.01</pc:scale>
  </pc:dimension>
  <pc:dimension>
    <pc:position>3</pc:position>
    <pc:size>4</pc:size>
    <pc:description>Z coordinate as a long integer. You must use the
                    scale and offset information of the header to
                    determine the double value.</pc:description>
    <pc:name>Z</pc:name>
    <pc:interpretation>int32_t</pc:interpretation>
    <pc:scale>0.01</pc:scale>
  </pc:dimension>
  <pc:dimension>
    <pc:position>4</pc:position>
    <pc:size>2</pc:size>
```

```
    <pc:description>The intensity value is the integer representa
tion
                    of the pulse return magnitude. This value is
optional
                    and system specific. However, it should always
be
                    included if available.</pc:description>
    <pc:name>Intensity</pc:name>
    <pc:interpretation>uint16_t</pc:interpretation>
    <pc:scale>1</pc:scale>
  </pc:dimension>
  <pc:metadata>
    <Metadata name="compression">dimensional</Metadata>
  </pc:metadata>
</pc:PointCloudSchema>');
```

- 创建点云表

```
-- 使用pcpoint数据类型
CREATE TABLE points (
    id SERIAL PRIMARY KEY,
    pt PCPOINT(1)
);

-- 使用pcpatch数据类型
CREATE TABLE patches (
    id SERIAL PRIMARY KEY,
    pa PCPATCH(1)
);
```

- 插入pcpoint类型数据

```
INSERT INTO points (pt)
SELECT ST_MakePoint(1, ARRAY[x,y,z,intensity])
FROM (
  SELECT
  -127+a/100.0 AS x,
    45+a/100.0 AS y,
         1.0*a AS z,
          a/10 AS intensity
  FROM generate_series(1,100) AS a
) AS values;

SELECT ST_MakePoint(1, ARRAY[-127, 45, 124.0, 4.0]);
-------------------------
010100000064CEFFFF94110000703000000400

SELECT ST_AsText('010100000064CEFFFF94110000703000000400'::pcpoint);
-------------------------
{"pcid":1,"pt":[-127,45,124,4]}
```

- 插入pcpatch类型数据

```
INSERT INTO patches (pa)
SELECT ST_Patch(pt) FROM points GROUP BY id/10;

SELECT ST_AsText(ST_MakePatch(1, ARRAY[-126.99,45.01,1,0, -126.98,45
.02,2,0, -126.97,45.03,3,0]));
-------------------------
{"pcid":1,"pts":[
 [-126.99,45.01,1,0],[-126.98,45.02,2,0],[-126.97,45.03,3,0]
```

```
]}
```

- pcpatch属性平均值计算

```
SELECT ST_AsText(ST_PatchAvg(pa)) FROM patches WHERE id = 7;
------------------------
{"pcid":1,"pt":[-126.46,45.54,54.5,5]}
```

- 删除扩展

```
Drop extension ganos_pointcloud_geometry;
Drop extension ganos_pointcloud cascade;
```

SQL参考

详细SQL手册请参见 PointCloud SQL参考。

2.5 轨迹模型

轨迹数据是针对移动对象（Moving Feature）所记录的连续位置变化信息，例如车辆的轨迹、人的轨迹等。轨迹数据是一类典型的时空数据，分析和理解这些轨迹数据能帮助人们研究许多重要问题。

概述

Ganos Trajectory是对象关系型数据库PostgreSQL的一个扩展，提供了一组数据类型、函数和存储过程，帮助用户高效地管理、查询和分析时空轨迹数据。

重要说明

ganos trajectory 1.6 版本和1.0版本不兼容，如要从1.0版本升级到1.6版本，请联系阿里云技术支持。

快速入门

- 创建扩展

```
Create Extension Ganos_trajectory cascade;
```

- 轨迹的枚举类型

```
CREATE TYPE leaftype AS ENUM ('STPOINT', 'STPOLYGON');
```

- 创建轨迹表

```
Create Table traj_table (id integer, traj trajectory);
```

- 插入轨迹数据

```
insert into traj_table values (1, ST_MakeTrajectory('STPOINT'::
leaftype, st_geomfromtext('LINESTRING (114 35, 115 36, 116 37)',
4326), '[2010-01-01 14:30, 2010-01-01 15:30)'::tsrange, '{"leafcount
```

```
": 3,"attributes" : {"velocity" : {"type":"integer","length":4,"
nullable":false,"value":[120, 130, 140]},"accuracy":{"type":"integer
","length":4,"nullable":false,"value":[120, 130, 140]},"bearing":{"
type":"float","length":4,"nullable":false,"value":[120, 130, 140]},"
acceleration":{"type":"float","length":4,"nullable":false,"value":
[120, 130, 140]}}}')), (2, ST_MakeTrajectory('STPOINT'::leaftype,
st_geomfromtext('LINESTRING (114 35, 115 36, 116 37)', 4326), '2010-
01-01 14:30'::timestamp, '2010-01-01 15:30'::timestamp, '{"leafcount
": 3,"attributes" : {"velocity" : {"type":"integer","length":4,"
nullable":false,"value":[120, 130, 140]},"accuracy":{"type":"integer
","length":4,"nullable":false,"value":[120, 130, 140]},"bearing":{"
type":"float","length":4,"nullable":false,"value":[120, 130, 140]},"
acceleration":{"type":"float","length":4,"nullable":false,"value":
[120, 130, 140]}}}')), (3, ST_MakeTrajectory('STPOINT'::leaftype,
st_geomfromtext('LINESTRING (114 35, 115 36, 116 37)', 4326),ARRAY
['2010-01-01 14:30'::timestamp, '2010-01-01 15:00'::timestamp, '
2010-01-01 15:30'::timestamp], '{"leafcount": 3,"attributes" : {"
velocity" : {"type":"integer","length":4,"nullable":false,"value":[
120, 130, 140]},"accuracy":{"type":"integer","length":4,"nullable":
false,"value":[120, 130, 140]},"bearing":{"type":"float","length":4
,"nullable":false,"value":[120, 130, 140]},"acceleration":{"type":"
float","length":4,"nullable":false,"value":[120, 130, 140]}}}')), (4
, ST_MakeTrajectory('STPOINT'::leaftype, st_geomfromtext('LINESTRING
 (114 35, 115 36, 116 37)', 4326), '[2010-01-01 14:30, 2010-01-01 15
:30)'::tsrange, null));
```

- **创建轨迹空间索引**

```
--创建基于函数的空间索引，加速空间过滤
create index tr_spatial_geometry_index on trajtab using gist (
st_trajectoryspatial(traj));

--空间查询时，加速空间过滤
select id, traj_id from traj_test where st_intersects(st_traject
oryspatial(traj), ST_GeomFromText('POLYGON((116.46747851805917 39
.92317964155052,116.4986540687358 39.92317964155052,116.4986540687
358 39.94452401711516,116.46747851805917 39.94452401711516,116.
46747851805917 39.92317964155052))')) = true and st_intersects(
st_trajectoryspatial(traj), ST_GeomFromText('POLYGON((116.5172424485
498 39.904984732832744,116.5543342491403 39.904984732832744,116.
5543342491403 39.93294918082651,116.5172424485498 39.93294918082651,
116.5172424485498 39.904984732832744))')) = true;
```

- **创建轨迹时间索引**

```
--创建基于函数的时间段索引，加速时间过滤
create index tr_timespan_time_index on trajtab using gist (
st_timespan(traj));

--创建基于函数的轨迹起止时间
create index tr_starttime_index on trajtab using btree (st_starttime
(traj));
create index tr_endtime_index on trajtab using btree (st_endtime(
traj));

--查询时，加速时间过滤
```

```
select id, traj_id from traj_split where st_starttime(traj) > '2008-
02-02 13:30:44'::timestamp and st_endtime(traj) < '2008-02-03 17:30:
44'::timestamp;
```

- **创建轨迹时间+空间复合索引(btree_gist)**

  btree_gist支持时间+空间的复合索引，适合轨迹查询条件中既有时间过滤，又有空间过滤的情况，使得时空复合查询SQL语法简单，且查询效率高。

```
--建立btree_gist 起始时间、终止时间、空间复合索引
create index tr_traj_test_stm_etm_sp_index on traj_test using gist (
st_starttime(traj),st_endtime(traj),st_trajectoryspatial(traj));

--时空查询
select id,traj_id from traj_test where st_starttime(traj) > '2008
-02-02 13:30:44'::timestamp and st_endtime(traj) < '2008-02-03 17
:30:44'::timestamp and st_intersects(st_trajectoryspatial(traj),
ST_GeomFromText('POLYGON((116.46747851805917 39.92317964155052,116
.4986540687358 39.92317964155052,116.4986540687358 39.9445240171
1516,116.46747851805917 39.94452401711516,116.46747851805917 39.
92317964155052))')) = true and st_intersects(st_trajectoryspatial
(traj), ST_GeomFromText('POLYGON((116.5172424485498 39.9049847328
32744,116.5543342491403 39.904984732832744,116.5543342491403 39.
93294918082651,116.5172424485498 39.93294918082651,116.5172424485498
 39.904984732832744))')) = true;
```

- **查询轨迹起、止时间**

```
select st_startTime(traj), st_endTime(traj) from traj_table ;
    st_starttime     |     st_endtime
---------------------+---------------------
 2010-01-01 14:30:00 | 2010-01-01 15:30:00
 2010-01-01 14:30:00 | 2010-01-01 15:30:00
 2010-01-01 14:30:00 | 2010-01-01 15:30:00
 2010-01-01 14:30:00 | 2010-01-01 15:30:00
 2010-01-01 14:30:00 | 2010-01-01 15:30:00
 2010-01-01 11:30:00 | 2010-01-01 15:00:00
 2010-01-01 11:30:00 | 2010-01-01 15:00:00
 2010-01-01 11:30:00 | 2010-01-01 15:00:00
(8 rows)
```

- **轨迹查询**

```
--通过插值函数查询轨迹点的属性
Select ST_velocityAtTime(traj, '2010-01-01 12:45') from traj_table
where id > 5;
st_velocityattime
-------------------
5
5
4.16666666666667
(3 rows)
```

- **分析轨迹间的相近性**

```
postgres=# Select ST_euclideanDistance((Select traj From traj_table
Where id = 6), (Select traj From traj_table Where id = 7));
 st_euclideandistance
----------------------
   0.0334968923954815
```

```
(1 row)
```

- **删除扩展**

```
Drop Extension Ganos_trajectory cascade;
```

SQL参考

- **详细SQL手册请参见 Trajectory SQL参考。**

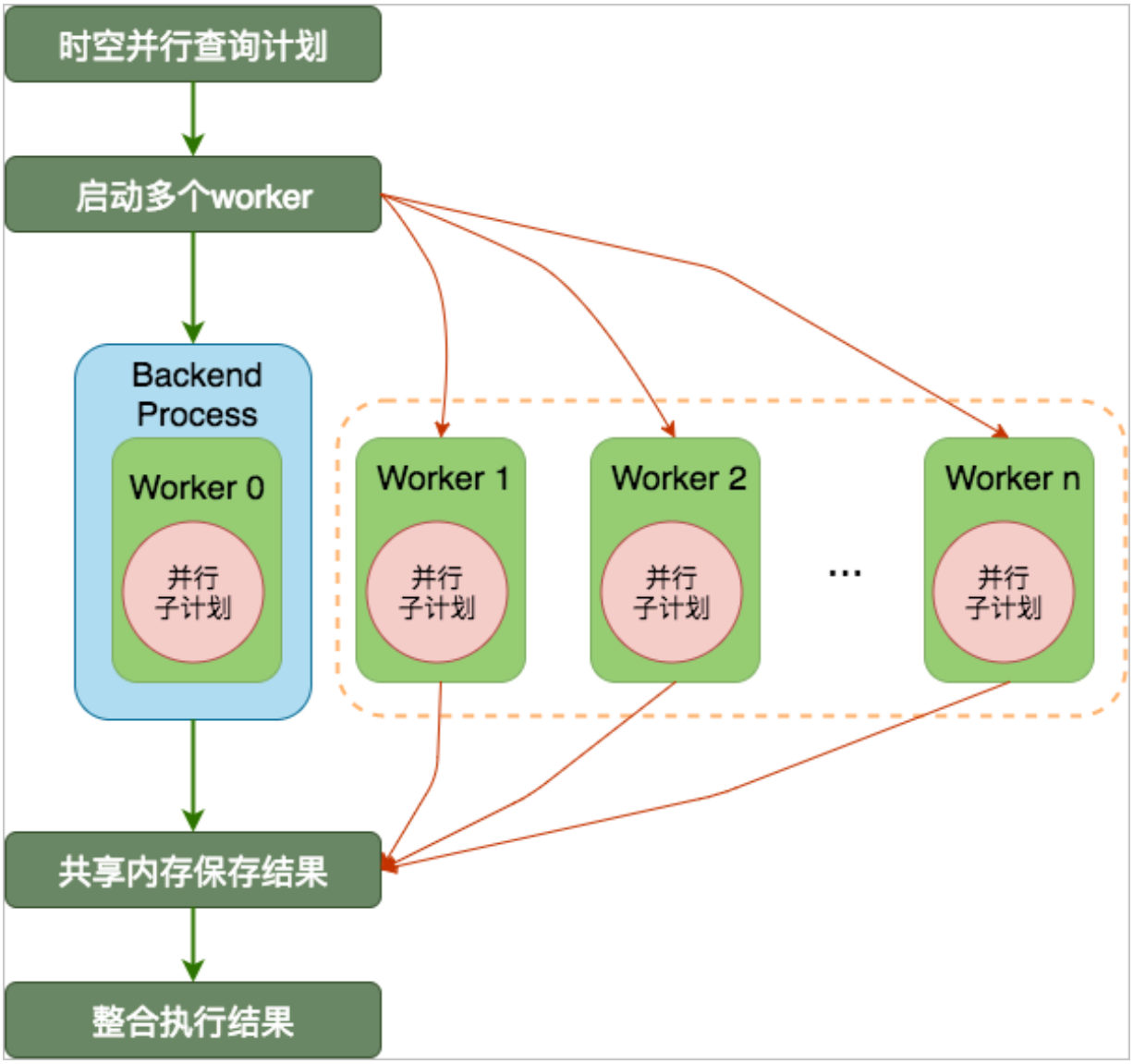
3 使用进阶

3.1 开启时空并行查询

对于大数据量、较复杂的时空查询，Ganos可直接利用PG并行查询的能力从而加速时空查询。

并行查询原理

PG并行查询是表级的并行，其并行查询示意图如下。



注意事项

- 并行查询的worker数量越大，worker数越多，查询时CPU负载越重，对于CPU负载本身较重的场景建议woker数量设置为2较合适，即max_parallel_workers_per_gather=2。

- 对于服务器内存有限的高并发访问，开启并行查询时，需要控制参数work_men（min 64KB），确保并发访问数量 乘以 并行worker数量 乘以 work_mem不超过服务器内存的60%。

使用方法

开启Ganos并行查询的方法如下：

1. 修改PostgreSQL配置文件postgresql.conf，启用并行查询参数。
   - 开启max_parallel_workers参数，设置能够开启的并行worker总数量，须小于max_worker_processes的值，通常为8-32。
   - 开启max_parallel_workers_per_gather参数，设置单个查询gahter最大并行度，须小于max_parallel_workers的值，通常为2-4。
   - 如果要开启强制并行，须将force_parallel_mode设置为on。
   - 通过执行sql语句控制单个表的并行粒度：alter table table_name set (parallel_workers=n)。

     
     说明:
     "n"代表并行worker数，该值可参考max_parallel_workers_per_gather。
2. 提高Ganos相关函数的cost成本。

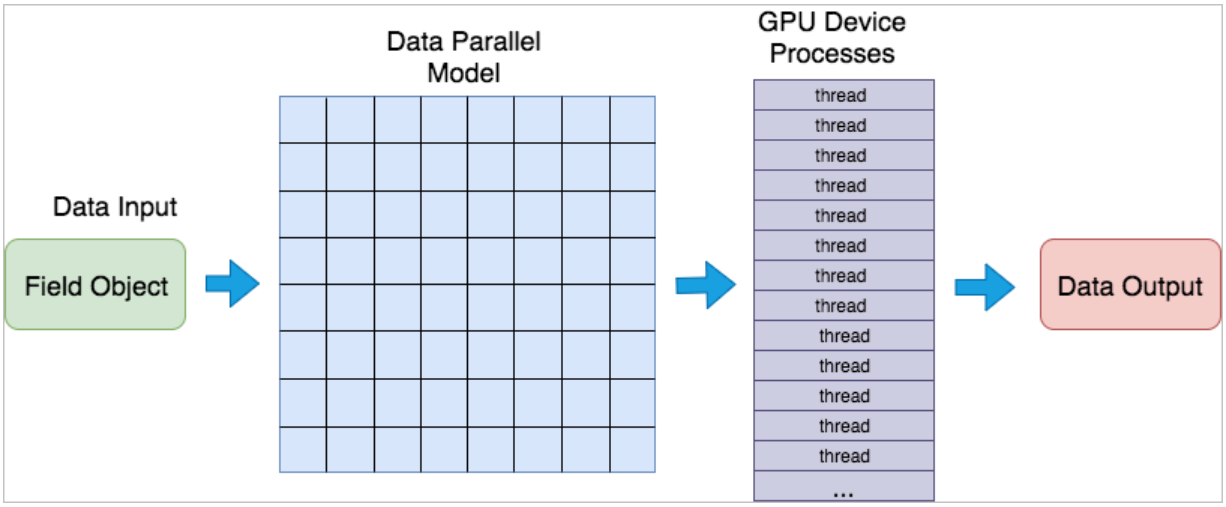
   在创建Ganos模块扩展时，通常默认有个函数cost成本，如果表数据量较小，但函数属于计算密集，并且适合开启并行执行，此时默认不会开启并行查询，需要提高函数的cost成本后才能开启并行查询。

3.2 开启GPU加速计算

GPU由于其特殊的硬件架构，在处理计算密集型、易于并行的程序上较CPU有很大的优势。

加速原理

数据库中GPU并行加速是指对象级的并行，将单个字段的对象转换为适合并行计算的模型，利用GPU超多核心的能力并行计算，其并行计算示意图如下。



注意事项

对于并发数较大的场景，单个GPU设备会存在资源受限的情况，所以建议在会话中关闭GPU加速计算功能。

使用方法

Ganos是通过检测GPU设备自动开启GPU加速计算，在实现函数上没有额外的参数设置，加速计算做到用户透明无感知；同时Ganos提供会话级控制权限，由用户决定是否启用GPU加速计算。

1. 确认当前环境是否有GPU设备。

a. 创建ganos_raster扩展"create extension ganos_raster cascade";

b. 执行sql语句"select st_checkgpu()"。

- **当前环境检测到GPU设备，成功返回GPU设备信息。**

```
rasterdb=# select st_checkgpu();
                    st_checkgpu

-------------------------------------------------------------------------------
 [GPU(0) prop]multiProcessorCount=20; sharedMemPerBlock=49152;
totalGlobalMem=-608239616; maxThreadsPerBlock=1024; maxThreads
PerMultiProcessor=2048; cudaThreadGetLimit=1024 .
(1 row)
```

- **当前环境未检测到GPU设备，则返回错误信息。**

```
rasterdb=# select st_checkgpu();
                   st_checkgpu

-------------------------------------------------------------------------------
There is not gpu device on current enviroment, cuda_errorcode
=35, errormsg=CUDA driver version is insufficient for CUDA
runtime version.
```

```
(1 row)
```

说明:

只有带GPU设备的环境才能启用GPU加速计算。

2. 开启和关闭会话级GPU使用状态。

   - 如果是带有GPU设备的环境，Ganos默认开启GPU加速计算，如果此时想关闭GPU加速计算，直接使用原来的CPU计算模式，则在会话中进行如下操作：

     执行set ganos.raster.use_cuda=off

```
rasterdb=# set ganos.raster.use_cuda=off;
SET
rasterdb=# show ganos.raster.use_cuda;
 ganos.raster.use_cuda
-----------------------
 off
(1 row)
```

   - 如果要重新开启GPU加速计算，执行set ganos.raster.use_cuda=on

```
rasterdb=# set ganos.raster.use_cuda=on;
SET
rasterdb=# show ganos.raster.use_cuda;
 ganos.raster.use_cuda
-----------------------
 on
(1 row)
```

3. Ganos中实现GPU加速计算的模块。

说明:

目前GPU加速计算仅在Ganos的Raster模块中应用实现，后续会增加Trajectory、Geometry模块的应用实现。

4 Raster SQL参考

4.1 基本概念

本章节将为您介绍Raster SQL的基本概念。

名称	描述
raster object	简称raster，栅格对象，是将空间分割成有规律的网格，每一个网格称为一个单元，并在各单元上赋予相应的属性值来表示实体的一种数据形式，可以是一幅卫星影像、一幅DEM或者一张图片。
cell/pixel	简称cell，栅格像元，也称栅格像素，即栅格对象中的一个网格，可以拥有不同的数据类型如Byte、Short、Int、Double等。
band	栅格波段，指栅格对象中的一层像元值矩阵，栅格对象可以有多个波段。
chunk	栅格块对象，块的大小可以自定义，比如256*256*3。
pyramid	栅格金字塔，是原始栅格对象的缩减采样版本，可以包含多个缩减采样图层，金字塔的各个连续图层均以2：1的比例进行缩减采样，第0层代表原始数据。
pyramid level	栅格金字塔层级。
mosaic	栅格镶嵌，将多个输入栅格镶嵌到现有栅格数据集。
interleaving	栅格像素交错方式，包括BSQ、BIP、BIL三种。
world space	世界坐标空间，即栅格对象的地理坐标空间。
raster space	栅格坐标空间，即栅格对象的像素坐标空间。
metadata	栅格的存储元数据（空间范围、投影类型、像素类型等），不包含遥感平台元数据。

4.2 Raster创建

4.2.1 ST_CreateRast

创建一个基于阿里云对象存储服务（OSS）的raster对象。

语法

```
raster ST_CreateRast(cstring url);
```

参数

参数名称	描述
url	OSS影像文件的路径。

描述

oss文件路径格式如下：`oss://access_id:secrect_key@endpoint/path_to/file`。其中endpoint可以被省略，系统会自动寻找相应的endpoint。如果endpoint被省略，路径必须以/开头。

Endpoint为oss的地域节点。为保证数据导入的性能，请确保云数据库PostgreSQL与OSS所在Region相同，相关信息请参考OSS endpoint信息。

示例

```
-- 基于OSS存储，指定accessID,accessKEY,endpoint
Select ST_CreateRast('OSS://ABCDEFG:1234567890@oss-cn.aliyuncs.com/
mybucket/data/4.tif');

-- 基于OSS存储，指定accessID,accessKEY
Select ST_CreateRast('OSS://ABCDEFG:1234567890@/mybucket/data/4.tif');

-- 基于OSS存储，直接指定https形式的URL，该模式性能较差
Select ST_CreateRast('https://mybuckets.oss-cn.aliyuncs.com/data/4.tif
');
```

4.3 导入导出

4.3.1 ST_ImportFrom

从一个OSS文件导入到数据库。

语法

```
raster ST_ImportFrom(cstring chunkTableName, cstring url);
```

参数

参数名称	描述
chunkTableName	块表的名称，名称必须符合数据库表名的规范。
url	外部文件路径，参考#unique_23中构建路径的描述。

描述

函数将创建一个raster对象，并将外部OSS文件导入到该对象中。

示例

```
Select ST_ImportFrom('chunk_table','OSS://ABCDEFG:1234567890@oss-cn.
aliyuncs.com/mybucket/data/4.tif');
```

4.3.2 ST_ExportTo

将一个raster对象导出为OSS文件。

语法

```
boolean ST_ExportTo(raster source,  cstring format, cstring url,
integer level = 0);
```

参数

参数名称	描述
source	需要导出的raster对象。
format	导出的数据，常见如 GTiff，BMP 等。
url	外部文件路径，参考#unique_25 中构建路径的描述。
level	金字塔级别。

描述

导出成功返回true，失败则返回false。

format指定导出格式的名称，常见格式如下。

名称	全称
BMP	Microsoft Windows Device Independent Bitmap(.bmp)
ECW	ERDAS Compressed Wavelets (.ecw)
EHdr	ESRI .hdr Labelled
GIF	Graphics InterchangeFormat(.gif)
GPKG	GeoPackage
GTiff	TIFF/BigTIFF/GeoTIFF(.tif)
HDF4	Hierarchical Data Format Release 4 (HDF4)
PDF	Geospatial PDF
PNG	Portable Network Graphics (.png)

示例

```
Select ST_ExportTo(raster, 'GTiff', 'OSS://ABCDEFG:1234567890@oss-cn.
aliyuncs.com/mybucket/data/4.tif') from raster_table where id=1;
```

4.4 金字塔操作

4.4.1 ST_BuildPyramid

创建影像金字塔。

语法

```
raster ST_BuildPyramid(raster source);
raster ST_BuildPyramid(raster source,  cstring chunkTableName);
```

参数

参数名称	描述
source	**需要创建金字塔的raster对象。**
chunkTableName	**金字塔所存储的分块表名称。**

描述

创建金字塔支持GPU加速，如果运行环境带有GPU设备，则Ganos会自动开启GPU加速功能。

示例

```
Update raster_table set raster_obj = ST_BuildPyramid(raster_obj) where
 id = 1;
```

```
Update raster_table set raster_obj = ST_BuildPyramid(raster_obj, '
chunk_table') where id = 2;
```

4.4.2 ST_deletePyramid

删除影像金字塔。

语法

```
raster ST_deletePyramid(raster source);
```

参数

参数名称	描述
source	需要删除金字塔的raster对象。

描述

删除影像金字塔，重置影像元数据，删除金字塔块数据。

示例

```
Update raster_table set raster_obj = ST_deletePyram id(raster_obj)
where id = 1;
```

4.4.3 ST_BestPyramidLevel

根据视口的世界坐标范围，长和宽来计算最佳的金字塔层级。

语法

```
integer ST_BestPyramidLevel(raster rast, Box extent, integer width,
integer height );
```

参数

参数名称	描述
rast	需要转换的raster对象。
box	视口的世界空间坐标范围，格式为((minX,minY),(maxX,maxY))。
width	视口的像素宽度。
height	视口的像素高度。

描述

raster对象必须要有完整的空间参考信息（srid值有效）。

示例

```
Select ST_BestPyramidLevel(raster_obj, '((128.0, 30.0),(128.5, 30.5
))', 800, 600) from raster_table where id = 10;

----------------------
3
```

4.5 坐标系统转换

4.5.1 ST_Rast2WorldCoord

由像元坐标及像元所在金字塔层级，根据仿射变换公式计算世界坐标。

语法

```
point ST_Rast2WorldCoord(raster raster_obj, integer pyramidLevel,
integer row, integer column);
```

参数

参数名称	描述
raster_obj	目标raster对象
pyramidLevel	金字塔层级
row	行号
column	列号

描述

raster对象必须要有完整的空间参考信息（srid值有效）。

示例

```
Select ST_Rast2WorldCoord(raster_obj, 0, 0, 0) from raster_table;
```

4.5.2 ST_World2RastCoord

由世界坐标及像元所在金字塔层级，根据逆仿射变换公式计算像元坐标。

语法

```
point ST_World2RastCoord(raster raster_obj,  integer pyramidLevel,
point coord);
```

参数

参数名称	描述
raster_obj	需要转换的raster对象。
pyramidLevel	需要转换的金字塔层级。
coord	需要转换的世界空间坐标。

描述

raster对象必须要有完整的空间参考信息（srid值有效）。

示例

```
Select ST_World2RastCoord(raster_obj, 0, '(27.9,128.6)') from
raster_table;
```

4.6 像素值操作

4.6.1 ST_ClipDimension

计算Clip结果的像素坐标。

语法

```
box ST_ClipDimension(raster raster_obj,  integer pyramidLevel,  box
extent);
```

参数

参数名称	描述
raster_obj	需要转换的raster对象。
pyramidLevel	需要转换的金字塔层级。

参数名称	描述
box	需要转换的世界空间坐标。

描述

raster对象必须要有完整的空间参考信息（srid值有效）。

示例

```
Select ST_ClipDimension(raster_obj, 2, '((128.0, 30.0),(128.5, 30.5
))') from raster_table where id = 10;

-------------------------------------
'((200, 300),(600, 720))'
```

4.6.2 ST_Clip

对raster对象进行裁剪操作。

语法

```
bytea ST_Clip(raster raster_obj,integer pyramidLevel,  box extent,
BoxType boxType);
bytea ST_Clip(raster raster_obj,integer pyramidLevel,  box extent,
BoxType boxType, integer destSrid);
record ST_Clip(raster raster_obj,
                      geometry geom,
                      integer pyramidLevel default 0,
                      cstring bands default '',  /* All bands */
                      float8[] nodata default NULL,
                      cstring clipOption default '',
                      cstring storageOption default '',
                      out box outwindow,
                      out bytea rasterblob)
```

参数

参数名称	描述
raster_obj	需要裁剪的raster对象。
pyramidLevel	金字塔层级。
extent	需要裁剪的范围，格式为'((minX,minY),(maxX,maxY))'。
boxType	范围的类型，只能是以下一种： · Raster （像元坐标） · World （世界坐标）
destSrid	指定的输出像元子集的空间参考值。
geometry	需要裁剪的geometry对象。

参数名称	描述
bands	需要裁剪的波段，用'0-2'或者‘1,2,3’ 这种形式表示，以0开始。 默认为'', 表示裁剪所有的波段。
nodata	用float8[]表示的nodata数值。 如果数值个数少于波段数量，则使用波段设置的nodata值填充。如果波段未设置nodata，则用0填充。
clipOption	json字符串表示的裁剪选项。
storageOption	json字符串表示的返回结果的存储选项。

clipOption参数如下。

参数名称	类型	默认值	描述
window_clip	bool	false	是否使用geometry的外包框进行裁剪。取值： · true：使用geometry的MBR裁剪； · false：使用geometry对象裁剪。

storageOption参数如下。

参数名称	类型	默认值	描述
compression	string	lz4	压缩算法类型。取值： · none · jpeg · zlib · png · lzo · lz4
quality	integer	75	压缩质量，只针对jpeg压缩算法。
interleaving	string	和原始raster一致	交错方式。取值： · bip：Band interleaved by pixel； · bil：Band nterleaved by pixel； · bsq：Band Sequential。
endian	string	和原始raster一致	字节序。取值： · NDR：Little endian； · XDR：Big endian。

描述

默认的裁剪缓存为100MB，代表最多只能裁剪出100MB大小的结果数据，如果需要调整返回结果大小，可使用参数 ganos.raster.clip_max_buffer_size 设置缓存的大小。

示例

```
Select ST_Clip(raster_obj, 0, '((128.980,30.0),(129.0,30.2))', 'World
');
Select ST_Clip(raster_obj, 0, '((128.980,30.0),(129.0,30.2))', 'World
',  4326);

--  使用geometry裁剪
-- 都是用默认值裁剪
SELECT (ST_CLIP(rast, ST_geomfromtext('Polygon((0 0, 45 45, 90 45, 45
0, 0 0))', 4326), 0)).* from clip_table where id =1

-- 使用白色作为背景色填充并且压缩为png图片
SELECT (ST_CLIP(rast, ST_geomfromtext('Polygon((0 0, 45 45, 90 45, 45
 0, 0 0))', 4326), 0, '', ARRAY[254,254,254], '', '{"compression":"png
","interleaving":"bip"}')).* from clip_table where id =1;

-- 使用geometry的窗口裁剪
SELECT (ST_CLIP(rast, ST_geomfromtext('Polygon((0 0, 45 45, 90 45, 45
 0, 0 0))', 4326), 0, '', NULL, '{"window_clip":true}', '')).* from
clip_table where id =1;
```

4.6.3 ST_ClipToRast

用指定的Geometry对象去裁剪Raster对象，并将裁剪结果作为一个新的Raster对象返回。

语法

```
raster ST_ClipToRast(raster raster_obj,
                     geometry geom,
                     integer pyramidLevel default 0,
                     cstring bands default '',  /* All bands */
                     float8[] nodata default NULL,
                     cstring clipOption default '',
                     cstring storageOption default '')
```

参数

参数名称	描述
raster_obj	需要裁剪的raster对象。
pyramidLevel	金字塔层级。
geometry	用于裁剪的geometry对象。
bands	需要裁剪的波段，用'0-2'或者‘1,2,3’ 这种形式表示，以0开始。 默认为'', 表示裁剪所有的波段。
nodata	用float8[]表示的nodata数值。 如果数值个数少于波段数量，则使用波段设置的nodata值填充。如果波段未设置nodata，则用0填充。

参数名称	描述
clipOption	json字符串表示的裁剪选项。
storageOption	json字符串表示的返回结果的存储选项。

clipOption参数如下。

参数名称	类型	默认值	描述
window_clip	bool	false	是否使用geometry的外包框进行裁剪。取值： · true：使用geometry的MBR裁剪； · false：使用geometry对象裁剪。

storageOption参数如下。

参数名称	类型	默认值	描述
chunking	boolean	和原始raster一致	是否使用分块存储。
chunkdim	string	和原始raster一致	分块的维度信息。在chunking=true时才有效。
chunktable	string	''	分块表名称。如果传入''值，则会产生一个随机表名临时块表用于存放数据。 该临时表只在当前会话中中有效。如果需要保持一个可访问的裁剪对象，则需要指定块表名称。
compression	string	lz4	压缩算法类型。取值： · none · jpeg · zlib · png · lzo · lz4
quality	integer	75	压缩质量，只针对jpeg压缩算法。
interleaving	string	和原始raster一致	交错方式。取值： · bip：Band interleaved by pixel； · bil：Band nterleaved by pixel； · bsq：Band Sequential。

参数名称	类型	默认值	描述
endian	string	和原始raster一致	字节序。取值： · NDR：Little endian； · XDR：Big endian。

描述

- 如果chunkTable传入为NULL或者"，则会产生一个随机表名的临时块表用于存放数据，该临时表只在当前会话中中有效。如果需要保持一个可访问的裁剪对象，则需要指定块表名称。
- 默认的裁剪缓存为100MB，代表最多只能裁剪出100MB大小的结果数据，如果需要调整返回结果大小，可使用参数 ganos.raster.clip_max_buffer_size 设置缓存的大小。

示例

```
-- 永久表
CREATE TEMP TABLE rast_clip_result(id integer, rast raster);
-- 临时表
CREATE TEMP TABLE rast_clip_result_temp(id integer, rast raster);

-- 默认裁剪并存放到临时表中
INSERT INTO rast_clip_result_temp(id, rast)
select 1, ST_ClipToRast(rast, ST_geomfromtext('Polygon((0 0, 45 45, 90
 45, 45 0, 0 0))', 4326), 0)
from clip_table
where id =1;

-- 使用白色作为背景色填充,并保存到永久表中
INSERT INTO rast_clip_result(id, rast)
select 2, ST_ClipToRast(rast, ST_geomfromtext('Polygon((0 0, 45 45, 90
 45, 45 0, 0 0))', 4326), 0, '', ARRAY[254,254,254], '', '{"chunktable
":"clip_rbt"}')
from clip_table
where id =1;

-- 使用geometry的窗口裁剪
INSERT INTO rast_clip_result_temp(id, rast)
SELECT 3, ST_ClipToRast(rast, ST_geomfromtext('Polygon((0 0, 45 45, 90
 45, 45 0, 0 0))', 4326), 0, '', NULL, '{"window_clip":true}', '')
from clip_table
where id =1;
```

4.6.4 ST_Values

查询Raster对象中与Geometry对象相交的所有像元对应的地理坐标以及像元值。

语法

```
set of record ST_Values(raster raster_obj,
                        geometry geom,
                        integer pyramidLevel default 0,
                        cstring bands default '',  /* All bands */
                        cstring clipOption default '',
                        out point coords,
                        out integer band,
```

```
                        out float8 value)
```

参数

参数名称	描述
raster_obj	需要裁剪的raster对象。
pyramidLevel	金字塔层级。
geometry	需要裁剪的geometry对象。
bands	需要裁剪的波段，用'0-2'或者‘1,2,3’ 这种形式表示，以0开始。 默认为'', 表示裁剪所有的波段。
clipOption	json字符串表示的裁剪选项。
point	返回结果字段之一，像素值的经纬度（地理）坐标。
band	返回结果字段之一，像素值所在波段号。
value	返回结果字段之一，像素值。

clipOption参数如下。

参数名称	类型	默认值	描述
window_clip	bool	false	是否使用geometry的外包框进行裁剪。取值： · true：使用geometry的MBR裁剪； · false：使用geometry对象裁剪。

描述

- Geometry对象与Raster对象都需要有空间参考信息，且两者空间参考必须一致。
- 默认的裁剪缓存为100MB，代表最多只能裁剪出100MB大小的结果数据，如果需要调整返回结果大小，可使用参数 ganos.raster.clip_max_buffer_size 调整缓存的大小。

示例

```
-- 基于点选择
SELECT ( ST_Values(rast, ST_geomfromtext('POINT(128.135 29.774)', 4326
)::geometry, 0,'{"window_clip":"true"}')).*
from rat_clip WHERE id=1;
    coord     | band | value
--------------+------+-------
 (127.8,29.7) |    0 |    11
 (127.8,29.7) |    1 |    10
 (127.8,29.7) |    2 |    50
(3 rows)

-- 基于线选择
SELECT ( ST_Values(rast, ST_geomfromtext('LINESTRING(0 0, 45 45, 90 0
, 135 45)', 4326)::geometry, 0)).*
from rat_clip WHERE id=1 limit 10;
```

```
    coord    | band | value
-------------+------+-------
 (44.1,45)   |    0 |   115
 (44.1,45)   |    1 |   112
 (44.1,45)   |    2 |    69
 (45,45)     |    0 |   122
 (45,45)     |    1 |   117
 (45,45)     |    2 |    75
 (134.1,45)  |    0 |    37
 (134.1,45)  |    1 |    64
 (134.1,45)  |    2 |    13
 (43.2,44.1) |    0 |    66
(10 rows)
```

4.6.5 ST_Update

用源raster更新目标raster。

语法

```
raster ST_Update(raster source,  raster dest);
```

参数

参数名称	描述
source	**源raster对象。**
dest	**目标raster对象。**

示例

```
Update raster_table Set raster_obj=ST_Update(raster_obj, (Select
raster_obj from raster_table where id=2)) where id = 1 ;
```

4.6.6 ST_MosaicFrom

将指定的raster对象进行镶嵌操作，合并成为一个新的raster对象。

语法

```
raster ST_MosaicFrom(raster source[],  cstring chunkTableName);
```

参数

参数名称	描述
source	**需要拼接的源raster对象。**
chunkTableName	**拼接完成的块表名称，必须符合数据库表名规范。**

描述

镶嵌函数会创建一个新的raster对象。

所有指定的raster对象需要满足以下条件：

- 具有相同的波段数。
- 所有的raster对象要么都进行了地理参考，要么都不是。如果都是地理参考，则采用世界坐标镶嵌。
- 指定raster对象的像素类型可以不同。如果是世界坐标镶嵌，则SRID、仿射参数必须一致。

示例

```
Insert Into raster_obj Values(1, ST_MosaicFrom(Array(select raster_obj
 from raster_table where id < 10), 'chunk_table_mosaic'))
Update raster_table Set raster_obj = ST_MosaicFrom(Array(select
raster_obj from raster_table where id < 10), 'chunk_table_mosaic')
where id = 11;
```

4.6.7 ST_MosaicTo

将raster对象或对象集镶嵌到目标raster对象

语法

```
raster ST_MosaicTo(raster raster_obj,raster source[]);
```

参数

参数名称	描述
raster_obj	目标raster对象。
source	源raster对象或对象集。

描述

源raster对象和目标raster对象需要满足以下条件：

- 具有相同的波段数。
- 所有的raster对象要么都进行了地理参考，要么都不是。如果都是地理参考，则采用世界坐标镶嵌。
- 指定raster对象的像素类型可以不同。如果是世界坐标镶嵌，则SRID、仿射参数必须一致。

示例

```
Update raster_table Set raster_obj = ST_MosaicTo(raster_obj, Array(
select raster_obj from raster_table where id < 10)) where id = 11;
```

4.7 Overview操作

4.7.1 ST_BuildOverview

创建一个overview(概视图)。

语法

```
raster ST_BuildOverview(cstring srcTableName, cstring srcColumnName,
integer srcPyramidLevel, cstring chunkTableName);
```

参数

参数名称	描述
tableName	表名称。
columnName	raster 列名称。
pyramidLevel	需要创建的overview的金字塔层级。
chunkTableName	生成的raster对象的块表名称。

函数支持创建一个基于表的raster overview对象。

所有涉及操作的raster对象需要满足以下条件：

- 具有相同的波段数。
- 所有的raster对象要么都进行了地理参考，要么都不是。如果都是地理参考，则采用世界坐标镶嵌。
- 指定raster对象的像素类型可以不同。如果是世界坐标镶嵌，则SRID、像素分辨率必须一致。

示例

```
Insert into raster_table_overview values(1, ST_BuildOverview('
raster_table','raster_obj', 0,'chunk_table_overview'));
```

4.7.2 ST_UpdateOverview

使用raster对象或对象集更新overview。

语法

```
raster ST_UpdateOverview(raster raster_obj,raster source[]);
```

参数

参数名称	描述
raster_obj	目标raster对象。
source	源raster对象或对象集。

描述

所有指定的raster对象需要满足以下条件：

- 具有相同的波段数。
- 所有的raster对象都有地理参考坐标，或者都没有，不允许存在部分有部分没有地理参考坐标的情况。如果都有地理参考坐标，则采用世界坐标（地理坐标）镶嵌。
- 指定raster对象的像素类型可以不同。如果是世界坐标镶嵌，则SRID、像素分辨率必须一致。

示例

```
Update raster_table set raster_obj = ST_UpdateOverview(raster_obj,
Array(select raster_obj from raster_table_new)) where id = 1;
```

4.7.3 ST_EraseOverview

清空一个raster对象的指定区域。

语法

```
raster ST_EraseOverview(raster raster_obj, Box extent, BoxType type,
boolean useNodata);
```

参数

参数名称	描述
raster_obj	需要操作的影像。
extent	需要清空的区域，格式为'((minX,minY),(maxX,maxY))'。
type	区域的类型，范围的类型，只能是以下一种：Raster（影像坐标）、World（世界坐标）。
useNodata	是否使用nodata值进行填充。如果指定为false或者未设置nodata值，则使用 0 进行填充。

示例

```
Select ST_EraseOverview(raster_obj, '((0,0),(100,100))', 'Raster',
false)  from raster_table where id=100;
```

4.8 DEM操作

4.8.1 ST_Aspect

计算DEM坡向，返回坡向数组。

语法

```
float8[] ST_Aspect(raster rast, integer pyramid_level, integer band,
Box extent, BoxType type);
```

参数

参数名称	描述
rast	raster对象。
pyramid_level	计算的金字塔等级。
Band	波段索引号。
box	分析区域，格式为'((m inX,m inY), (m axX,m axY))'。
type	分析区域的坐标类型，只能是以下一种： · Raster（影像坐标） · World（世界坐标）

描述

坡向用于识别出从每个像元到其相邻像元方向上值的变化率最大的下坡方向。坡向可以被视为坡度方向。输出栅格中各像元的值可指示出各像元位置处表面的朝向的罗盘方向。将按照顺时针方向进行测量，角度范围介于0（正北）到360（仍是正北）之间，即完整的圆。不具有下坡方向的平坦区域将赋值为-1。

坡向数据集中每个像元的值都可指示出该像元的坡度朝向。

示例

```
select st_aspect(rast, 0, 0, '(0,0), (5,5)', 'Raster') from t_surface
where id=1;
                                                        st_aspect

-------------------------------------------------------------------------------------
 {5.68600864908691,5.742765806909,5.08963804002705,5.2696845155022,5.
67764064357462,5.91019758537953,5.37643514321137,5.
..48370356795928,5.18761908299415,5.02882043600982,4.83181790640303,4.
33940125858463,5.56909460856743,5.92299216276176,.
.5.86712080145576,5.29376916418959,4.41982721398084,4.53158940708072,0
.141897054604164,0.274167451119659,6.267561578559.
.11,5.742765806909,4.56734728205808,4.7215630349298,0.547562235939998,
6.0814229628593,5.96909743805248,5.80535425218747.
.,4.74463586281994,4.71238898038469,0.141897054604164,6.13281387917245
,6.05997388768217,5.98920538937637,4.789160871654.
.47,4.4855901323308}
```

```
(1 row)
```

4.8.2 ST_Slope

计算坡度，返回弧度为单位的像元坡度数组。

语法

```
float8[] ST_Slope(raster rast, integer pyramid_level, integer band,
Box extent, BoxType type, float8 zfactor);
```

参数

参数名称	描述
rast	**raster对象。**
pyramid_level	**计算的金字塔等级。**
Band	**波段索引号**
box	**分析区域，格式为'((m inX,m inY), (m axX,m axY))'。**
type	**分析区域的坐标类型，只能是以下一种：** **· Raster（影像坐标）** **· World（世界坐标）**
zfactor	**高程夸张值，默认为1。**

描述

坡度计算函数用于为每个像元计算值在从该像元到与其相邻的像元方向上的最大变化率。实际上，高程随着像元与其相邻的八个像元之间距离的变化而产生的最大变化率，可用来识别自该像元开 始的最陡坡降。

示例

```
select st_slope(rast, 0, 0, '(0,0), (5,5)', 'Raster', 2.0) from
t_surface where id=1;
                                                          st_slope

------------------------------------------------------------------------------
 {0.210279822382945,0.369954478614613,0.220241089748741,0.4155048857
24335,0.523380429142649,0.19079849696355,0.32493941.
.6771785,0.592806538904308,0.559435506670953,0.487598684366856,0.
17107200555711,0.0840217922621739,0.381860307736563,0..
.580949493078414,0.638382145824945,0.43822706997925,0.317039337499457,
0.284095352866283,0.284298114561498,0.49448931974.
.7884,0.817870125632039,0.645927342349833,0.241209216517688,0.
211549813235801,0.339040463954188,0.636582806346833,0.934.
.672430381381,0.7814534477193,0.0832677675316725,0.0544326656785603,0.
529537557031012,0.836305912538514,0.9446346707062.
.92,0.747858756227042,0.0284106287186713,0.0616861154423774}
```

```
(1 row)
```

4.8.3 ST_Hillshade

计算山影，返回山影数组。

语法

```
float8[] ST_Hillshade(raster rast, integer pyramid_level, integer band
, Box extent, BoxType type, float8 zfactor, float8 azimuth, float8
altitude);
```

参数

参数名称	描述
rast	raster对象。
pyramid_level	计算的金字塔等级。
band	波段索引号。
extent	分析区域，格式为'((m inX,m inY), (m axX,m axY))'。
type	分析区域的坐标类型，只能是以下一种： · Raster（影像坐标） · World（世界坐标）
zfactor	高程夸张值，默认为1。
azimuth	太阳方位角，默认为315（西北）顺时针，范围为0-360。
altitude	太阳高度角，太阳在正方为90，范围为0-90。

描述

山体阴影函数通过为栅格中的每个像元确定照明度，来获取表面的假定照明度。通过设置假定光源的位置和计算与相邻像元相关的每个像元的照明度值，即可得出假定照明度。进行分析或图形显示时，特别是使用透明度时，“山体阴影”工具可大大增强表面的可视化。

默认情况下，阴影和光线是与介于0和255之间的整数相关的灰度梯度（从黑色渐变为白色）。

示例

```
select st_hillshade(rast, 0, 0, '(0,0), (5,5)', 'Raster', 4, 180, 80)
from t_surface where id=1;
                                                    st_hillshade

------------------------------------------------------------------------------
 {241.058488938393,213.471315764272,248.503699937536,219.598763873892,
186.13564444109,240.161828059936,230.539211856565.
.,181.175441801687,200.382310292847,219.449103772262,254.328476494409,
254.486010192554,216.118274956217,165.59445864749.
```

```
.,156.73816546518,215.145437887027,241.33319958436,247.721999074798,
213.697052503065,162.469277526395,107.176240164182,.
.160.555287495495,251.337972845835,254.741770272405,193.116915680406,
148.402860944697,101.093528491212,133.798418058854.
.,254.285971266744,253.163140286752,158.495693029955,109.654658826363,
95.2639231161562,131.371235276504,252.85982399282.
.2,254.364747782334}
(1 row)
```

4.8.4 ST_Overflow

根据区域和区域中的降水量输出区域内代表水深度的像元值。

语法

```
float8[] ST_Overflow(raster rast, Box box, float8 value);
```

参数

参数名称	描述
rast	raster对象，目前仅支持1波段的DEM。
box	分析区域，世界坐标（与DEM一致）。
value	分析区域的水量（立方米）。

描述

根据区域和区域中的降水量输出区域内代表水深度的像元值，raster对象目前仅支持1波段的DEM数据。

示例

```
select st_overflow(rast, '((-195516.0,2230052.0),(-194626.0,2230942.0
))'::box, 10000::float8) from t_overflow where id =1;

                          st_overflow



 ----------------------------------------------------------------------------------
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1.43427103409668,0,0.
434271034096678,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.
434271034096678,0,0.434271034096678,0,0.434271034096678,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.434271034096678,0.4342710340
96678,0.434271034096678,0.434271034096678,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.434271034096678,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.434271034096678,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.434271034096678,0,0,0,0,0,0,0
```

```
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.434271034096678,0.
434271034096678,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0.434271034096678,0.434271034096678,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.434271034096678,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.434271034096678,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.
434271034096678,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,1.43427103409668,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.434271034096678,0,0,0,0,0}
(1 row)
```

4.8.5 ST_Flow_direction

根据区域中的DEM高程值，计算区域中的水流方向。

语法

```
float8[] ST_flow_direction(raster rast, Box box);
```

参数

参数名称	描述
rast	**raster对象，目前仅支持1波段的DEM。**
box	**分析区域，世界坐标。**

描述

使用d8算法，根据区域中的DEM高程值，计算区域中的水流方向。

raster对象目前仅支持1波段的DEM数据。

示例

```
select st_flow_direction(rast, '((-202286.94,2232375.16),(-202135.0,
2232225))'::box, 1000::float8) from t_overflow where id =2;
                               st_flow_direction

------------------------------------------------------------------------------
 {2,2,2,4,4,8,2,2,2,4,4,8,1,1,2,4,8,4,128,128,1,2,4,8,2,2,1,4,4,4,1,1,
1,1,4,16}
(1 row)
```

4.9 属性的查询与更新

4.9.1 ST_Name

获得raster对象的名称。如果没有定义名称，则返回空值。

语法

```
text ST_Name(raster rast);
```

参数

参数名称	描述
rast	**raster对象。**

示例

```
select ST_Name(rast) from rat where id=1;

_________________________________
image1
```

4.9.2 ST_SetName

设置raster对象的名称。

语法

```
raster ST_SetName(raster rast, cstring name);
```

参数

参数名称	描述
rast	**raster对象。**
name	**新的对象名称。**

示例

```
update rat set rast = ST_SetName(rast,'image2') where id = 2;

_____________________________________
```

```
(1 row)
```

4.9.3 ST_MetaData

获得raster对象的元数据，返回json格式。

语法

```
text ST_MetaData(raster raster_obj);
```

参数

参数名称	描述
raster_obj	**raster对象。**

示例

```
select ST_MetaData(raster_obj) from raster_table;
```

4.9.4 ST_Width

获得raster对象的宽度。如果需要获得分块的宽度，参见ST_ChuckWidth。

语法

```
integer ST_Width(raster raster_obj);
```

参数

参数名称	描述
raster_obj	**raster对象。**

示例

```
select ST_Width(raster_obj) from raster_table;
_____________________________
```

```
10060
```

4.9.5 ST_Height

获得raster对象的高度。

语法

```
integer ST_Height(raster raster_obj);
```

参数

参数名称	描述
raster_obj	**raster对象.**

示例

```
select ST_Height(raster_obj) from raster_table;
```

4.9.6 ST_NumBands

获得raster对象的波段数。

语法

```
integer ST_NumBands(raster raster_obj);
```

参数

参数名称	描述
raster_obj	**raster对象。**

示例

```
select ST_NumBands(raster_obj) from raster_table;
_____________________________________
```

```
3
```

4.9.7 ST_Value

输出指定波段和行列号的像元值。

语法

```
float8 ST_Value(raster rast, integer band, integer colsn, integer
rowsn, boolean exclude_nodata_value);
```

参数

参数名称	描述
rast	**raster对象。**
band	**波段序号。**
colsn	**像元列号。**
rowsn	**像元行号。**
exclude_no data_value	**是否排除nodata，默认值为true。**

描述

输出指定波段和行列号的像元值（波段索引号从0开始），波段号和行列号默认为0。

示例

```
select st_value(rast, 1, 3, 4) from t_pixel where id=2;
 st_value
----------
       88
(1 row)
```

4.9.8 ST_RasterID

获得raster对象的UUID（通用唯一识别码）。

语法

```
text ST_RasterID(raster raster_obj);
```

参数

参数名称	描述
raster_obj	**raster对象。**

示例

```
select ST_RasterID(raster_obj) from raster_table;

______________________________________
4e692ed0-74e2-42a3-a10d-c28d4ae31982
```

4.9.9 ST_CellDepth

获得raster对象的像素深度。深度值可以为以下值之一：0，1，2，4，8，16，32，64。其中，0为像素深度未知。

语法

```
integer ST_CellDepth(raster raster_obj);
```

参数

参数名称	描述
raster_obj	**raster对象。**

示例

```
select ST_CellDepth(raster_obj) from raster_table;

_________________________________
8
```

4.9.10 ST_CellType

获得raster对象的像素类型。类型值可以为以下值之一："8BSI"， "8BUI"， "16BSI"，"16BUI"， "32BSI"， "32BUI"， "32BF"， "64BF"。

语法

```
text ST_CellType(raster raster_obj);
```

参数

参数名称	描述
raster_obj	**raster对象。**

示例

```
select st_celltype(raster_obj) from raster_table;

_________________________________
```

```
8BUI
```

4.9.11 ST_InterleavingType

获得raster对象的交错类型。交错类型可以是以下其中的一种："BSQ"，"BIL"，"BIP"。

语法

```
text ST_InterleavingType(rasterraster_obj);
```

参数

参数名称	描述
raster_obj	**raster对象。**

示例

```
select ST_InterleavingType(raster_obj) from raster_table;

_____________________________________
BSQ
```

4.9.12 ST_TopPyramidLevel

获得raster对象的金字塔最高层级。

语法

```
integer TopPyramidLevel(raster raster_obj);
```

参数

参数名称	描述
raster_obj	**raster对象。**

示例

```
select ST_TopPyramidLevel(raster_obj) from raster_table;

_____________________________________
```

```
6
```

4.9.13 ST_Extent

获得raster对象的坐标范围，返回PostgreSQL的BOX对象，格式为'((minX,minY),(maxX,maxY))'。

语法

```
BOX ST_Extent(raster raster_obj,CoorSpatialOption csOption = '
WorldFirst')
```

参数

参数名称	描述
raster_obj	raster对象。
CoorSpatialOption	坐标空间选项枚举值。

描述

CoorSpatialOption为坐标空间选项，可取以下值：

- Raster：影像坐标空间，返回像元坐标；
- World：世界坐标空间，返回世界坐标；
- WorldFirst：世界坐标空间优先，即如果已地理参考，则返回世界坐标，如果未地理参考，则返回像元坐标。

示例

```
select ST_Extent(raster_obj, 'Raster') from raster_table;

___________________________________
(0 0, 255 255)
```

4.9.14 ST_Srid

获得raster对象的空间参考标识符。空间参考标识符(SRID)的标识以及定义保存在系统表spatial_ref_sys。

语法

```
integer ST_Srid(raster raster_obj);
```

参数

参数名称	描述
raster_obj	raster对象。

示例

```
select ST_Srid(raster_obj) from raster_table where id=1;

__________________________________
4326
```

4.9.15 ST_SetSrid

设置raster对象的空间参考标识符。空间参考标识符(SRID)的标识以及定义保存在系统表spatial_ref_sys。

语法

```
raster ST_SetSrid(raster rast, integer srid);
```

参数

参数名称	描述
rast	raster对象。
srid	指定的空间参考标识符。

描述

对于存储模式为External的数据，执行该操作时必须确定更新的栅格对象已被空间参考并且更新的SRID能在空间参考系统表中找到，否则更新无效。

示例

```
update rast set rast=ST_SetSrid(rast,4326) where id=1;
__________________________________
(1 row)
```

4.9.16 ST_Georeference

获得raster对象的地理参考信息。text格式的仿射参数为："A,B,C,D,E,F"。

语法

```
text ST_Georeference(raster raster_obj);
```

参数

参数名称	描述
raster_obj	raster对象。

示例

```
select ST_Georeference(raster_obj) from raster_table where id=1;

____________________________________
2.500000000000000,0.000000000000000,38604686.750000000000000,0.
000000000000000,-2.500000000000000,4573895.750000000000000
```

4.9.17 ST_IsGeoreferenced

获取raster对象是否已被地理参考。boolean格式的返回结果为：“t”或“f”。

语法

```
boolean ST_IsGeoreferenced(raster raster_obj);
```

参数

参数名称	描述
raster_obj	**raster对象。**

描述

返回结果t表示true，f表示false。

示例

```
select ST_IsGeoreferenced(raster_obj) from raster_table where id=1;

____________________________________
t
```

4.9.18 ST_UnGeoreference

去掉raster对象的地理参考信息。

语法

```
raster ST_UnGeoreference(raster rast);
```

参数

参数名称	描述
rast	**raster对象。**

示例

```
update rast set rast=ST_UnGeoreference(rast) where id=1;

____________________________________
```

```
(1 row)
```

4.9.19 ST_SetGeoreference

设置raster对象的地理参考信息。

语法

```
raster ST_SetGeoreference(raster rast, integer srid, integer aop,
double A,double B, double C, double D, double E, double F);
```

参数

参数名称	描述
rast	raster对象。
srid	指定的空间参考标识符。
aop	AOP为空间参考点，取像元的中心或左上角： · Center=1 · Upleft =2
A~F	A~F为仿射变换的六个参数： · x = A*col + B*row + C · y = D*col + E*row + F

描述

对于存储模式为External的数据，执行该操作时必须确定更新的栅格对象已被空间参考并且更新的SRID能在空间参考系统表中找到，否则更新无效。

示例

```
update rast set rast=ST_SetGeoreference(rast,4326,1,8.4163,0,124,0,-8.
4163,36.2) where id=1;


_________________________________
```

```
(1 row)
```

4.9.20 ST_NoData

获取raster对象的某一个波段NoData（无效值标识）的值。如果没有NODATA定义，则返回空值。

语法

```
float8 ST_NoData(raster raster_obj, integer band);
```

参数

参数名称	描述
raster_obj	**raster对象。**
band	**波段序号，从0开始。**

示例

```
select ST_NoData(raster_obj, 0) from raster_table where id=1;

__________________________________
0.000
```

4.9.21 ST_SetNoData

设置raster对象的指定波段的NoData（无效值标识）的值。

语法

```
raster ST_SetNoData(raster rast, integer band_sn, double nodata_value
);
```

参数

参数名称	描述
rast	**raster对象。**
band	**指定的波段序号，从0开始，-1表示所有波段。**
nodata_value	**指定设置的nodata值。**

示例

```
update rast set rast=ST_SetNoData(rast,0, 999.999);

__________________________________
```

```
(1 row)
```

4.9.22 ST_ColorTable

获取raster对象的某一个波段的颜色表信息，返回颜色表的json格式。

语法

```
text ST_ColorTable(raster raster_obj, integer band);
```

参数

参数名称	描述
raster_obj	**raster对象。**
band	**指定的波段序号，从0开始。**

描述

颜色表的json格式：

- **4分量：**

```
'{"compsCount":4,
    "entries":[
        {"value":0,"c1":0,"c2":0,"c3":0,"c4":255},
        {"value":1,"c1":0,"c2":0,"c3":85,"c4":255},
        {"value":2,"c1":0,"c2":0,"c3":170,"c4":255}
    ]
}'
```

- **3分量：**

```
'{"compsCount":3,
    "entries":[
        {"value":0,"c1":0,"c2":0,"c3":0},
        {"value":1,"c1":0,"c2":0,"c3":85},
        {"value":2,"c1":0,"c2":0,"c3":170}
    ]
}'
```

如果不存在颜色表，函数返回空值。

示例

```
select ST_ColorTable(raster_obj,0) from raster_table where id = 1;

-----------------------------------
'{"compsCount":3,
    "entries":
    [
        {"value":0,"c1":0,"c2":0,"c3":0},
        {"value":1,"c1":0,"c2":0,"c3":85},
        {"value":2,"c1":0,"c2":0,"c3":170}
    ]
```

```
}'
```

4.9.23 ST_SetColorTable

设置raster对象的指定波段的颜色表信息，采用颜色表的json格式。

语法

```
raster ST_SetColorTable(raster rast, integer band_sn, cstring clb);
```

参数

参数名称	描述
rast	raster对象。
band_sn	指定的波段序号，从0开始。
clb	颜色表的json格式,参考ST_ColorTable。

示例

```
update rast set rast=ST_SetColorTable(rast,0,
'{"compsCount":4,
    "entries":[
        {"value":0,"c1":0,"c2":0,"c3":0,"c4":255},
        {"value":1,"c1":0,"c2":0,"c3":85,"c4":255},
        {"value":2,"c1":0,"c2":0,"c3":170,"c4":255}
    ]
}');

_________________________________
(1 row)
```

4.9.24 ST_Statistics

获取raster对象的某一个波段的统计值信息的json格式。如果不存在统计值，则返回空值。

语法

```
text ST_Statistics(raster raster_obj, integer band);
```

参数

参数名称	描述
raster_obj	Raster对象。
band	波段序号，从0开始。

示例

```
select ST_Statistics(raster_obj, 0) from raster_table where id=1;
```

```
_________________________________
'{ "min" : 0.00, "max" : 255.00,"mean" : 125.00,"std" : 23.123,"approx
" : false}'
```

4.9.25 ST_SetStatistics

设置raster对象的指定波段的统计值信息。

语法

```
raster ST_SetStatistics(raster rast, integer band, double min, double
max, double mean, double std,cstring samplingParams);
```

参数

参数名称	描述
rast	**raster对象。**
band	**指定的波段序号，从0开始。**
min,max,mean,std	**统计值。**
samplingParams	**'approx=false' 或者 'appro x=true'。**

示例

```
update rast set rast=ST_SetStatistics(rast,0,0.0 , 255.0, 125.0, 23.6,
 'approx=false');

_________________________________
(1 row)
```

4.9.26 ST_SummaryStats

计算一个raster对象的指定波段集的统计值信息。

语法

```
raster ST_SummaryStats(raster raster_obj) ;
```

参数

参数名称	描述
raster_obj	**Raster对象。**

示例

```
update raster_obj set raster_obj=ST_SummaryStats(raster_obj) where id
 = 1;
_________________________________
```

```
(1 row)
```

4.9.27 ST_ColorInterp

获取raster对象的某一个波段的颜色解释类型。

语法

```
text ST_ColorInterp(raster raster_obj, integer band);
```

参数

参数名称	描述
raster_obj	**Raster对象。**
band	**波段序号，从0开始。**

返回的interp值及其说明如下表。

值	说明
Undefined	**颜色解释类型未定义。**
GrayIndex	**灰度值索引。**
PaletteIndex	**颜色表索引。**
RedBand	**RGB颜色模型Red波段。**
GreenBand	**RGB颜色模型Green波段。**
BlueBand	**RGB颜色模型Blue波段。**
AlphaBand	**RGBA颜色模型Alpha波段。**
HueBand	**HSL颜色模型中Hue波段。**
SaturationBand	**HSL颜色模型中Saturation波段。**
LightnessBand	**HSL颜色模型中Lightness波段。**
CyanBand	**CMYK颜色模型中Cyan波段。**
MagentaBand	**CMYK颜色模型中Magenta波段。**
YellowBand	**CMYK颜色模型中Yellow波段。**
BlackBand	**CMYK颜色模型中Black波段。**
YCbCr_YBand	**YCBCR颜色模型中Y波段。**
YCbCr_CbBand	**YCBCR颜色模型中Cb波段。**
YCbCr_CrBand	**YCBCR颜色模型中Cr波段。**

示例

```
select ST_ColorInterp(raster_obj,0) from raster_table where id = 1;

_____________________________________
RedBand
```

4.9.28 ST_SetColorInterp

设置raster对象的指定波段的颜色解释类型。

语法

```
raster ST_SetColorInterp(raster rast, integer band_sn, ColorInterp
interp);
```

参数

参数名称	描述
rast	raster对象。
band_sn	指定的波段序号，从0开始。
interp	interp枚举值。

描述

interp枚举值及其解释：

值	说明
Undefined	颜色解释类型未定义。
GrayIndex	关联灰度颜色表。
RGBIndex	关联RGB颜色表。
RGBAIndex	关联RGBA颜色表。
CMYKIndex	关联CMYK颜色表。
HSLIndex	关联HSL颜色表。
RedBand	红色波段。
GreenBand	绿色波段。
BlueBand	蓝色波段。
AlphaBand	透明波段。
HueBand	HLS的色调分量。
SaturationBand	HLS的饱和度分量。

值	说明
LightnessBand	HLS的亮度分量。
CyanBand	CMYK的青色波段。
MegentaBand	CMYK的品红波段。
YellowBand	CMYK的黄色波段。
BlackBand	CMYK的黑色波段。
YBand	YCBCR的亮度分量。
CbBand	YCBCR的蓝色色度分量。
CrBand	YCBCR的红色色度分量。

示例

```
update rast set rast=ST_SetColorInterp(rast,0, 'CI_Cyan');

_________________________________
(1 row)
```

4.9.29 ST_Histogram

获取raster对象的指定波段的统计直方图信息，以文本格式返回。如果不存在直方图，函数返回空值。

语法

```
text ST_Histogram(raster raster_obj, integer band);
```

参数

参数名称	描述
raster_obj	Raster对象。
band	波段序号，从0开始。

示例

```
select ST_Histogram(raster_obj, 0) from raster_table where id=1;
_________________________________
{
 "approximate":false,
 "histsCounts":
 [2,1,1,0,8,17,47,101,193,345,443,640,877,1189,1560,1847,2087,2560,
2816,3193,3567,3840,4101,4415,4498,3876,3235,2458,1800,1598,1087,731,
638,426,264,198,147,126,104,104,80,84,86,71,80,62,74,85,72,80,70,88,69
,68,62,58,63,51,53,55,54,56,55,63,47,39,49,59,66,62,64,73,66,72,67,84
,86,79,91,92,117,138,136,142,157,225,287,285,382,449,567,628,750,855,
1021,1142,1242,1410,1504,1590,1786,1870,2044,2099,2277,2373,2451,2585
```

```
,2646,2882,2878,3091,3396,3620,3911,4124,4304,4700,4893,5314,5446,5657
,5765,5649,5749,5753,5601,5335,5161,4943,4592,4445,4207,4083,4090,4270
,4465,4514,4844,5204,5331,5597,5777,5838,6004,6316,6095,5762,5567,5465
,4923,4677,4220,3843,3401,3041,2571,2345,1972,1725,1376,1140,1008,841,
716,548,442,373,308,212,133,78,68,31,24,12,2,2,1],
 "binFunction":
 {
  "type":"unknown",
  "binRange":
  {
   "minValue":29.0,
   "maxValue":208.0,
   "outRange":"include",
   "binValues":
   [29.0,30.0,31.0,32.0,33.0,34.0,35.0,36.0,37.0,38.0,39.0,40.0,41.0,
42.0,43.0,44.0,45.0,46.0,47.0,48.0,49.0,50.0,51.0,52.0,53.0,54.0,55.0
,56.0,57.0,58.0,59.0,60.0,61.0,62.0,63.0,64.0,65.0,66.0,67.0,68.0,69.0
,70.0,71.0,72.0,73.0,74.0,75.0,76.0,77.0,78.0,79.0,80.0,81.0,82.0,83.0
,84.0,85.0,86.0,87.0,88.0,89.0,90.0,91.0,92.0,93.0,94.0,95.0,96.0,97.0
,98.0,99.0,100.0,101.0,102.0,103.0,104.0,105.0,106.0,107.0,108.0,109.0
,110.0,111.0,112.0,113.0,114.0,115.0,116.0,117.0,118.0,119.0,120.0,121
.0,122.0,123.0,124.0,125.0,126.0,127.0,128.0,129.0,130.0,131.0,132.0,
133.0,134.0,135.0,136.0,137.0,138.0,139.0,140.0,141.0,142.0,143.0,144
.0,145.0,146.0,147.0,148.0,149.0,150.0,151.0,152.0,153.0,154.0,155.0,
156.0,157.0,158.0,159.0,160.0,161.0,162.0,163.0,164.0,165.0,166.0,167
.0,168.0,169.0,170.0,171.0,172.0,173.0,174.0,175.0,176.0,177.0,178.0,
179.0,180.0,181.0,182.0,183.0,184.0,185.0,186.0,187.0,188.0,189.0,190
.0,191.0,192.0,193.0,194.0,195.0,196.0,197.0,198.0,199.0,200.0,201.0,
202.0,203.0,204.0,205.0,206.0,207.0]
  }
 }
}
```

4.9.30 ST_SetHistogram

设置raster对象的指定波段的直方图信息，参数采用JSON文本格式进行定义。

语法

```
raster ST_SetHistogram(raster rast, integer band, cstring histogram);
```

参数

参数名称	描述
rast	**raster对象。**
band	**指定的波段序号，从0开始。**
histogram	**基于JSON描述的直方图。**

描述

histogram基于JSON格式描述，具体的定义如下。

参数名称	描述	类型	说明
approximate	**是否采用抽样方法**	**boolean**	-

参数名称	描述	类型	说明
histsCounts	数据量数组	integer[]	-
binFunction/type	所采用的bin函数类型	string	· linear为线性模型 · logarithm为对数模 型 · explicit为显式指定
binFunction/binTable/binValues	bin值	number[]	explicit有效
binFunction/binRange/minValue	最小值	number	logarithm linear有效
binFunction/binRange/maxValue	最大值	number	logarithm linear有效
binFunction/binRange/outRange	超出值	number	logarithm linear有效
binFunction/binRange/binValues	bin值	number[]	logarithm linear有效

```
eg.1:
{
  "approximate":false,
  "histsCounts":
  [2,1,1,0,8,17,47,101,193,345,443,640,877,1189,1560,1847,2087,2560,
2816,3193,3567,3840,4101,4415,4498,3876,3235,2458,1800,1598,1087,731,
638,426,264,198,147,126,104,104,80,84,86,71,80,62,74,85,72,80,70,88,69
,68,62,58,63,51,53,55,54,56,55,63,47,39,49,59,66,62,64,73,66,72,67,84
,86,79,91,92,117,138,136,142,157,225,287,285,382,449,567,628,750,855,
1021,1142,1242,1410,1504,1590,1786,1870,2044,2099,2277,2373,2451,2585
,2646,2882,2878,3091,3396,3620,3911,4124,4304,4700,4893,5314,5446,5657
,5765,5649,5749,5753,5601,5335,5161,4943,4592,4445,4207,4083,4090,4270
,4465,4514,4844,5204,5331,5597,5777,5838,6004,6316,6095,5762,5567,5465
,4923,4677,4220,3843,3401,3041,2571,2345,1972,1725,1376,1140,1008,841,
716,548,442,373,308,212,133,78,68,31,24,12,2,2,1],
  "binFunction":
  {
    "type":"unknown",
    "binRange":
    {
      "minValue":29.0,
      "maxValue":208.0,
      "outRange":"include",
      "binValues":
      [29.0,30.0,31.0,32.0,33.0,34.0,35.0,36.0,37.0,38.0,39.0,40.0,41.
0,42.0,43.0,44.0,45.0,46.0,47.0,48.0,49.0,50.0,51.0,52.0,53.0,54.0,55.
0,56.0,57.0,58.0,59.0,60.0,61.0,62.0,63.0,64.0,65.0,66.0,67.0,68.0,69.
```

```
0,70.0,71.0,72.0,73.0,74.0,75.0,76.0,77.0,78.0,79.0,80.0,81.0,82.0,83.
0,84.0,85.0,86.0,87.0,88.0,89.0,90.0,91.0,92.0,93.0,94.0,95.0,96.0,97.
0,98.0,99.0,100.0,101.0,102.0,103.0,104.0,105.0,106.0,107.0,108.0,109
.0,110.0,111.0,112.0,113.0,114.0,115.0,116.0,117.0,118.0,119.0,120.0,
121.0,122.0,123.0,124.0,125.0,126.0,127.0,128.0,129.0,130.0,131.0,132
.0,133.0,134.0,135.0,136.0,137.0,138.0,139.0,140.0,141.0,142.0,143.0,
144.0,145.0,146.0,147.0,148.0,149.0,150.0,151.0,152.0,153.0,154.0,155
.0,156.0,157.0,158.0,159.0,160.0,161.0,162.0,163.0,164.0,165.0,166.0,
167.0,168.0,169.0,170.0,171.0,172.0,173.0,174.0,175.0,176.0,177.0,178
.0,179.0,180.0,181.0,182.0,183.0,184.0,185.0,186.0,187.0,188.0,189.0,
190.0,191.0,192.0,193.0,194.0,195.0,196.0,197.0,198.0,199.0,200.0,201.
0,202.0,203.0,204.0,205.0,206.0,207.0]
    }
  }
}

eg.2:
{
  "approximate" : true,
  "histsCounts" :[1,2,3,4,5],
  "binFunction" :
  {
    "type" : "explicit",
    "binTable" :
    {
      "binValues" : [1.0,2.0,3.0,4.0,5.0]
    }
  }
}
```

示例

```
UPDATE rat SET raster = st_sethistogram(raster, 0, '{"approximate
":true,"histsCounts":[1,2,3,4,5],"binFunction":{"type":"explicit","
binTable":{"binValues":[1.0,2.0,3.0,4.0,5.0]}}}') where id =1;

---------------------------------
(1 row)
```

4.9.31 ST_BuildHistogram

计算一个raster对象的指定波段集的直方图信息。

语法

```
raster ST_BuildHistogram(raster raster_obj);
```

参数

参数名称	描述
raster_obj	Raster对象。

示例

```
UPDATE raster_table SET raster_obj = st_buildhistogram(raster_obj)
WHERE id = 1;
---------------------------------
```

```
(1 row)
```

4.10 辅助函数

4.10.1 ST_CheckGPU

验证是否有GPU环境。

语法

```
text  ST_CheckGPU();
```

描述

验证当前数据库运行环境是否有可识别的GPU硬件设备。

示例

```
select st_checkgpu();

------------------------------------------------------------------------------------
 [GPU prop]multiProcessorCount=20; sharedMemPerBlock=49152; maxThreads
PerBlock=1024

(1 row)
```

5 PointCloud SQL参考

5.1 构造函数

5.1.1 ST_makePoint

构造一个pcpoint对象。

语法

```
pcpoint ST_makePoint(integer pcid, float8[] vals);
```

参数

参数名称	描述
pcid	schema的id，来自表point_cloud_formats。
float8[]	float8数组，数组元素个数取决于schema的dimension。

示例

```
SELECT ST_makePoint(1, ARRAY[-127, 45, 124.0, 4.0]);
-----------------------------------------------
010100000064CEFFFF94110000703000000400
```

5.1.2 ST_makePatch

构造一个pcpatch对象。

语法

```
pcpatch ST_makePatch(integer pcid,  float8[] vals);
```

参数

参数名称	描述
pcid	schema的id，来自表point_cloud_formats。
float8[]	float8数组，数组元素为schema的dimension维度数的整数倍。

示例

```
SELECT ST_asText(ST_MakePatch(1, ARRAY[-126.99,45.01,1,0, -126.98,45.
02,2,0, -126.97,45.03,3,0]));
-----------------------------------------------
{"pcid":1,"pts":[
 [-126.99,45.01,1,0],[-126.98,45.02,2,0],[-126.97,45.03,3,0]
]}
```

5.1.3 ST_Patch

通过pcpoint数组构造一个pcpatch对象。

语法

```
pcpatch ST_Patch(pcpoint[]  pts);
```

参数

参数名称	描述
pts	**pcpoint数组。**

示例

```
INSERT INTO patches (pa)
SELECT ST_Patch(pt) FROM points GROUP BY id/10;
```

5.2 属性函数

5.2.1 ST_asText

将pcpoint/pcpatch对象转为json字符串表达。

语法

```
text ST_asText(pcpatch pp);
text ST_asText(pcpoint pt);
```

参数

参数名称	描述
pp	**pcpatch对象。**
pt	**pcpoint对象。**

示例

```
SELECT ST_asText('010100000064CEFFFF94110000703000000400'::pcpoint);
-------------------------------
```

```
{"pcid":1,"pt":[-127,45,124,4]}
```

5.2.2 ST_pcID

获取pcpoint/pcpatch对象的schema ID。

语法

```
integer ST_pcID(pcpoint pt);
integer ST_pcID(pcpatch pp);
```

参数

参数名称	描述
pt	**pcpoint对象。**
pp	**pcpatch对象。**

示例

```
SELECT ST_pcID('010100000064CEFFFF94110000703000000400'::pcpoint);
--------------------------------
1
```

5.2.3 ST_get

获取pcpoint对象属性值。

语法

```
float8[]  ST_get(pcpoint pc);
numeric ST_get(pcpoint pc, text dimname);
```

参数

参数名称	描述
pc	**pcpoint对象。**
dimname	**指定的属性维度名称。**

示例

```
SELECT ST_Get('010100000064CEFFFF94110000703000000400'::pcpoint, '
Intensity');
--------------------
4

SELECT ST_Get('010100000064CEFFFF94110000703000000400'::pcpoint);
--------------------
```

```
{-127,45,124,4}
```

5.2.4 ST_numPoints

获取pcpatch对象中的pcpoint个数。

语法

```
integer ST_numPoints(pcpatch pc);
```

参数

参数名称	描述
pc	**pcpatch对象。**

示例

```
SELECT ST_NumPoints(pa) FROM patches LIMIT 1;
---------------------------
9
```

5.2.5 ST_summary

获取pcpatch对象的概要信息。

语法

```
text ST_summary(pcpatch pc);
```

参数

参数名称	描述
pc	**pcpatch对象。**

示例

```
SELECT ST_Summary(pa) FROM patches LIMIT 1;
----------------------------------
{"pcid":1, "npts":9, "srid":4326, "compr":"dimensional","dims":[{"pos
":0,"name":"X","size":4,"type":"int32_t","compr":"sigbits","stats":{"
min":-126.99,"max":-126.91,"avg":-126.95}},{"pos":1,"name":"Y","size":
4,"type":"int32_t","compr":"sigbits","stats":{"min":45.01,"max":45.09
,"avg":45.05}},{"pos":2,"name":"Z","size":4,"type":"int32_t","compr":"
sigbits","stats":{"min":1,"max":9,"avg":5}},{"pos":3,"name":"Intensity
","size":2,"type":"uint16_t","compr":"rle","stats":{"min":0,"max":0,"
avg":0}}]}
```

5.3 对象操作

5.3.1 ST_compress

将pcpatch对象按指定方式压缩。

语法

```
pcpatch ST_compress(pcpatch pc, text global_compression_schema default
 '', text compression_config default '');
```

参数

参数名称	描述
pc	pcpatch对象。
global_compression_schema	压缩框架。
compression_config	压缩配置项，指定具体维度的压缩算法。

描述

压缩框架可以为：

```
auto            -- determined by pcid
dimension
laz              -- no compression config supported
ght             -- is discarded
```

当压缩框架为dimension时，压缩配置项可以为：

```
auto -- determined automatically, from values stats
zlib -- deflate compression
sigbits -- significant bits removal
rle -- run-length encoding
```

示例

```
SELECT ST_asText(ST_Compress(ST_MakePatch(1, ARRAY[-126.99,45.01,1,0,
 -126.98,45.02,2,0, -126.97,45.03,3,0])));

------------------------------------------------
{"pcid":1,"pts":[
 [-126.99,45.01,1,0],[-126.98,45.02,2,0],[-126.97,45.03,3,0]
```

```
]}
```

5.3.2 ST_unCompress

将pcpatch对象解压。

语法

```
pcpatch ST_unCompress(pcpatch pc);
```

参数

参数名称	描述
pc	pcpatch对象。

描述

所有返回pcpatch对象的接口，在返回pcpatch前都是先按schema中指定的压缩方法进行了压缩。

示例

```
SELECT ST_Uncompress(pa) FROM patches  WHERE ST_NumPoints(pa) = 1;
---------------------------------------
0101000000000000000001000000C8CEFFFFF81100000102700000A00
```

5.3.3 ST_union

将pcpatch数组聚合成单个pcpatch对象。

语法

```
pcpatch ST_union(pcpatch[] pcs);
```

参数

参数名称	描述
pcs	pcpatch数组。

示例

```
-- Compare npoints(sum(patches)) to sum(npoints(patches))
SELECT ST_NumPoints(ST_Union(pa)) FROM patches;
SELECT Sum(ST_NumPoints(pa)) FROM patches;
```

```
100
```

5.3.4 ST_explode

将pcpatch对象反解为多行的pcpoint。

语法

```
setof[pcpoint] ST_union(pcpatch pc);
```

参数

参数名称	描述
pc	pcpatch对象。

示例

```
SELECT ST_AsText(ST_Explode(pa)), id FROM patches WHERE id = 7;
-----------------------------------------
               st_astext                | id
----------------------------------------+----
 {"pcid":1,"pt":[-126.5,45.5,50,5]}     |  7
 {"pcid":1,"pt":[-126.49,45.51,51,5]}   |  7
 {"pcid":1,"pt":[-126.48,45.52,52,5]}   |  7
 {"pcid":1,"pt":[-126.47,45.53,53,5]}   |  7
 {"pcid":1,"pt":[-126.46,45.54,54,5]}   |  7
 {"pcid":1,"pt":[-126.45,45.55,55,5]}   |  7
 {"pcid":1,"pt":[-126.44,45.56,56,5]}   |  7
 {"pcid":1,"pt":[-126.43,45.57,57,5]}   |  7
 {"pcid":1,"pt":[-126.42,45.58,58,5]}   |  7
 {"pcid":1,"pt":[-126.41,45.59,59,5]}   |  7
```

5.3.5 ST_patchAvg

计算pcpatch对象中pcpoint所有属性的平均值，返回新的pcpoint对象。

语法

```
pcpoint ST_patchAvg(pcpatch pc);
```

参数

参数名称	描述
pc	pcpatch对象。

示例

```
SELECT ST_AsText(ST_PatchAvg(pa)) FROM patches WHERE id = 7;
-------------------------------------
```

```
{"pcid":1,"pt":[-126.46,45.54,54.5,5]}
```

5.3.6 ST_patchMax

计算pcpatch对象中pcpoint所有属性的最大值，返回新的pcpoint对象。

语法

```
pcpoint ST_patchMax(pcpatch pc);
```

参数

参数名称	描述
pc	**pcpatch对象。**

示例

```
SELECT ST_PatchMax(pa) FROM patches WHERE id = 7;
---------------------------------
{"pcid":1,"pt":[-126.41,45.59,59,5]}
```

5.3.7 ST_patchMin

计算pcpatch对象中pcpoint所有属性的最小值，返回新的pcpoint对象。

语法

```
numeric ST_patchAvg(pcpatch pc);
```

参数

参数名称	描述
pc	**pcpatch对象。**

示例

```
SELECT ST_PatchMin(pa) FROM patches WHERE id = 7;
---------------------------------
```

```
{"pcid":1,"pt":[-126.5,45.5,50,5]}
```

5.3.8 ST_patchAvg

计算pcpatch对象中所有pcpoint某一属性维度的平均值。

语法

```
numeric ST_patchAvg(pcpatch pc, text dimname);
```

参数

参数名称	描述
pc	**pcpatch对象。**
dimname	**指定的属性维名称。**

示例

```
SELECT ST_PatchAvg(pa, 'intensity') FROM patches WHERE id = 7;
--------------------------
5.0000000000000000
```

5.3.9 ST_patchMax

计算pcpatch对象中所有pcpoint某一属性维度的最大值。

语法

```
numeric ST_patchMax(pcpatch pc, text dimname);
```

参数

参数名称	描述
pc	**pcpatch对象。**
dimname	**指定的属性维名称。**

示例

```
SELECT ST_PatchMax(pa, 'intensity') FROM patches WHERE id = 7;
--------------------------
```

```
125.0000000000000000
```

5.3.10 ST_patchMin

计算pcpatch对象中所有pcpoint某一属性维度的最小值。

语法

```
numeric ST_patchAvg(pcpatch pc, text dimname);
```

参数

参数名称	描述
pc	**pcpatch对象。**
dimname	**指定的属性维名称。**

示例

```
SELECT ST_PatchMin(pa, 'intensity') FROM patches WHERE id = 7;
--------------------------
0.25122
```

5.3.11 ST_filterGreaterThan

指定pcpoint某一维度的固定值，过滤出pcpatch中所有该维度值大于该固定值的pcpoint，并以新的pcpatch对象返回。

语法

```
pcpatch ST_filterGreaterThan(pcpatch pc, text dimname, float8 value);
```

参数

参数名称	描述
pc	**pcpatch对象。**
dimname	**指定的属性维名称。**
value	**属性固定值。**

示例

```
SELECT ST_AsText(ST_FilterGreaterThan(pa, 'y', 45.57)) FROM patches
WHERE id = 7;
------------------------------------------------------------
```

```
 {"pcid":1,"pts":[[-126.42,45.58,58,5],[-126.41,45.59,59,5]]}
```

5.3.12 ST_filterLessThan

指定pcpoint某一维度的固定值，过滤出pcpatch中所有该维度值小于该固定值的pcpoint，并以新的pcpatch对象返回。

语法

```
pcpatch ST_filterLessThan(pcpatch pc, text dimname, float8 value);
```

参数

参数名称	描述
pc	**pcpatch对象。**
dimname	**指定的属性维名称。**
value	**属性固定值。**

示例

```
SELECT ST_AsText(ST_FilterLessThan(pa, 'y', 45.60)) FROM patches WHERE
 id = 7;
------------------------------------------------------------
 {"pcid":1,"pts":[[-126.42,45.58,58,5],[-126.41,45.59,59,5]]}
```

5.3.13 ST_filterEquals

指定pcpoint某一维度的固定值，过滤出pcpatch中所有该维度值等于该固定值的pcpoint，并以新的pcpatch对象返回。

语法

```
pcpatch ST_filterEquals(pcpatch pc, text dimname, float8 value);
```

参数

参数名称	描述
pc	**pcpatch对象。**
dimname	**指定的属性维名称。**
value	**属性固定值。**

示例

```
SELECT ST_AsText(ST_FilterEquals(pa, 'y', 45.57)) FROM patches WHERE
id = 7;
------------------------------------------------------------
```

```
 {"pcid":1,"pts":[[-126.42,45.57,58,5],[-126.41,45.57,59,5]]}
```

5.3.14 ST_filterBetween

指定pcpoint某一维度的一大一小两个固定值，过滤出pcpatch中所有该维度值处于两个固定值中间的pcpoint，并以新的pcpatch对象返回。

语法

```
pcpatch ST_filterBetween(pcpatch pc, text dimname, float8 minvalue,
float8 maxvalue);
```

参数

参数名称	描述
pc	pcpatch对象。
dimname	指定的属性维名称。
minvalue	属性固定值最小值。
maxvalue	属性固定值最大值。

描述

返回的结果中，不包括等于一大一小两个固定值的情况。

示例

```
SELECT ST_AsText(ST_FilterBetween(pa, 'y', 45.57, 45.60)) FROM patches
 WHERE id = 7;
------------------------------------------------------------
 {"pcid":1,"pts":[[-126.42,45.58,58,5],[-126.41,45.59,59,5]]}
```

5.3.15 ST_pointN

返回pcpatch中指定序号的pcpoint对象。

语法

```
pcpoint ST_pointN(pcpatch pc, integer n);
```

参数

参数名称	描述
pc	pcpatch对象。
n	指定的序号，从1开始，如果是负数，则从pcpatch的末尾开始往前推算|n|。例如n=-2，则从pacpatch的末尾往前推算2位。

示例

```
SELECT ST_asText(ST_pointN(pa, 4)) FROM patches WHERE id = 7;
-----------------------------------
{"pcid":1,"pt":[-126.41,45.59,59,5]}
```

5.3.16 ST_isSorted

判断pcpatch中所有pcpoint是否按指定的维度进行排序。

语法

```
boolean ST_isSorted(pcpatch pc, text[] dimnames, boolean strict
default true);
```

参数

参数名称	描述
pc	**pcpatch对象。**
dimnames	**属性维度名称数组。**
strict	**如果为true则进一步要求单个pcpoint的属性维度无重复。**

示例

```
SELECT ST_isSorted(pa, ['y','m']::array) FROM patches;
--------------------------    ----------------------------------
 f
(1 rows)
```

5.3.17 ST_sort

将pcpatch中所有pcpoint按指定的维度进行排序，返回新的pcpatch。

语法

```
pcpatch ST_sort(pcpatch pc, text[] dimnames);
```

参数

参数名称	描述
pc	**pcpatch对象。**
dimnames	**属性维度名称数组。**

示例

```
update patches set pa=ST_Sort(pa, ['y','m']::array);
-------------------------------------
```

```
(1 rows)
```

5.3.18 ST_range

获取pcpatch中从指定序号start开始后的n个pcpoint，返回新的pcpatch。

语法

```
pcpatch ST_range(pcpatch pc, integer start, integer n);
```

参数

参数名称	描述
pc	pcpatch对象。
start	指定的pcpoint序号，基数从1开始。
n	指定序号（包含自身）往后的pcpoint个数。

示例

```
update patches set pa=ST_range(pa, 2, 16);
--------------------------------------
(1 rows)
```

5.3.19 ST_setPcid

给pcpatch对象设置新的schema，返回新的pcpatch。

语法

```
pcpatch ST_setPcid(pcpatch pc, integer pcid, float8 def default 0.0);
```

参数

参数名称	描述
pc	pcpatch对象。
pcid	新的schema ID值，来自表pointcloud_formats。
def	指定值，针对在新的schema中存在而在旧的schema中不存在的属性维，设为该指定值，该指定值的默认值是0.0。

描述

在旧的schema中存在而新的schema中不存在的属性维将被丢弃。

示例

```
update patches set pa=ST_setPcid(pa, 2, 0.0);
---------------------------------------
(1 rows)
```

5.3.20 ST_transform

将pcpatch对象转换为新的schema，返回新的pcpatch。

语法

```
pcpatch ST_transform(pcpatch pc, integer pcid, float8 def default 0.0
);
```

参数

参数名称	描述
pc	**pcpatch对象。**
pcid	**新的schema ID值，来自表pointcloud_formats。**
def	**指定值，针对在新的schema中存在而在旧的schema中不存在的属性维，设为该指定值，该指定值的默认值是0.0。**

描述

与st_setpcid不同的地方是，st_transform允许将pcpatch的值根据新的schema中维度的重解读、缩放、偏移而进行改变。

示例

```
update patches set pa=ST_transform(pa, 2, 0.0);
---------------------------------------
(1 rows)
```

5.3.21 ST_envelopeGeometry

返回pcpatch对象外包框geometry。

语法

```
geometry ST_envelopeGeometry(pcpatch pc);
```

参数

参数名称	描述
pc	**pcpatch对象。**

描述

外包框为ganos geometry的2D geometry对象。

示例

```
SELECT ST_AsText(ST_EnvelopeGeometry(pa)) FROM patches LIMIT 1;
------------------------------
POLYGON((-126.99 45.01,-126.99 45.09,-126.91 45.09,-126.91 45.01,-126.
99 45.01))

CREATE INDEX ON patches USING GIST(ST_EnvelopeGeometry(patch));
```

5.3.22 ST_boundingDiagonalGeometry

返回pcpatch对象外包框对角线geometry。

语法

```
geometry ST_boundingDiagonalAsBinary(pcpatch pc);
```

参数

参数名称	描述
pc	pcpatch对象。

描述

外包框对角线为ganos geometry的2D LineString对象。本函数可用在pcpatch列建索引。

示例

```
SELECT ST_AsText(ST_BoundingDiagonalGeometry(pa)) FROM patches;
                  st_astext
-----------------------------------------------
LINESTRING Z (-126.99 45.01 1,-126.91 45.09 9)
LINESTRING Z (-126 46 100,-126 46 100)
LINESTRING Z (-126.2 45.8 80,-126.11 45.89 89)
LINESTRING Z (-126.4 45.6 60,-126.31 45.69 69)
LINESTRING Z (-126.3 45.7 70,-126.21 45.79 79)
LINESTRING Z (-126.8 45.2 20,-126.71 45.29 29)
LINESTRING Z (-126.5 45.5 50,-126.41 45.59 59)
LINESTRING Z (-126.6 45.4 40,-126.51 45.49 49)
LINESTRING Z (-126.9 45.1 10,-126.81 45.19 19)
LINESTRING Z (-126.7 45.3 30,-126.61 45.39 39)
LINESTRING Z (-126.1 45.9 90,-126.01 45.99 99)


CREATE INDEX ON patches USING GIST(ST_BoundingDiagonalGeometry(patch)
gist_geometry_ops_nd);
```

5.4 OGC WKB操作

5.4.1 ST_asBinary

返回pcpoint对象的wkb值。

语法

```
bytea ST_asBinary(pcpoint pt);
```

参数

参数名称	描述
pt	pcpoint对象。

示例

```
SELECT ST_AsBinary('010100000064CEFFFF941100007030000000400'::pcpoint);
-----------------------------------------------
\x0101000080000000000c05fc0000000000804640000000000005f40
```

5.4.2 ST_envelopeAsBinary

返回pcpatch对象外包框的wkb值。

语法

```
bytea ST_envelopeAsBinary(pcpatch pc);
```

参数

参数名称	描述
pc	pcpatch对象。

描述

外包框为ganos geometry的2D geometry对象。

示例

```
SELECT ST_EnvelopeAsBinary(pa) FROM patches LIMIT 1;
-----------------------------------------------
\x0103000000010000000500000090c2f5285cbf5fc0e17a
14ae4781464090c2f5285cbf5fc0ec51b81e858b46400ad7
a3703dba5fc0ec51b81e858b46400ad7a3703dba5fc0e17a
```

```
14ae4781464090c2f5285cbf5fc0e17a14ae47814640
```

5.4.3 ST_boundingDiagonalAsBinary

返回pcpatch对象外包框对角线的wkb值。

语法

```
bytea ST_boundingDiagonalAsBinary(pcpatch pc);
```

参数

参数名称	描述
pc	pcpatch对象。

描述

外包框为2D geometry。

示例

```
SELECT ST_BoundingDiagonalAsBinary( ST_Patch(ARRAY[ ST_MakePoint(1,
ARRAY[0.,0.,0.,10.]), ST_MakePoint(1, ARRAY[1.,1.,1.,10.]), ST_MakePoi
nt(1, ARRAY[10.,10.,10.,10.])]));
------------------------------------------------------------
\x01020000a0e610000002000000000000000000000000000000000000000000
0000000000000000000000000024400000000000002440000000000000244000
```

5.5 空间关系判断

5.5.1 ST_intersects

判断两个pcpatch对象的外包框是否相交。

语法

```
boolean ST_intersects(pcpatch pp1, pcpatch pp2);
boolean ST_intersects(geometry g,  pcpatch pp1);
```

参数

参数名称	描述
pp1	pcpatch对象1。
pp2	pcpatch对象2。
g	ganos geometry的geometry对象。

示例

```
-- Patch should intersect itself
SELECT ST_Intersects(
         '0101000000000000000001000000C8CEFFFFF81100001027000000A00'::
pcpatch,
         '0101000000000000000001000000C8CEFFFFF81100001027000000A00'::
pcpatch);
------------------------
t

SELECT ST_Intersects('SRID=4326;POINT(-126.451 45.552)'::geometry, pa
) FROM patches WHERE id = 7;
------------------------
t
```

5.6 空间处理

5.6.1 ST_intersection

将pcpatch对象和给定的几何对象进行相交处理，返回相交处理后的子pcpatch对象。

语法

```
pcpatch ST_intersection(pcpatch pc, geometry geom);
```

参数

参数名称	描述
pc	**pcpatch对象。**
geom	**ganos geometry的geometry对象。**

示例

```
SELECT ST_AsText(ST_Explode(ST_Intersection(
      pa,
      'SRID=4326;POLYGON((-126.451 45.552, -126.42 47.55, -126.40 45.
552, -126.451 45.552))'::geometry
)))
FROM patches WHERE id = 7;

             st_astext
-------------------------------------
 {"pcid":1,"pt":[-126.44,45.56,56,5]}
 {"pcid":1,"pt":[-126.43,45.57,57,5]}
 {"pcid":1,"pt":[-126.42,45.58,58,5]}
 {"pcid":1,"pt":[-126.41,45.59,59,5]}
```

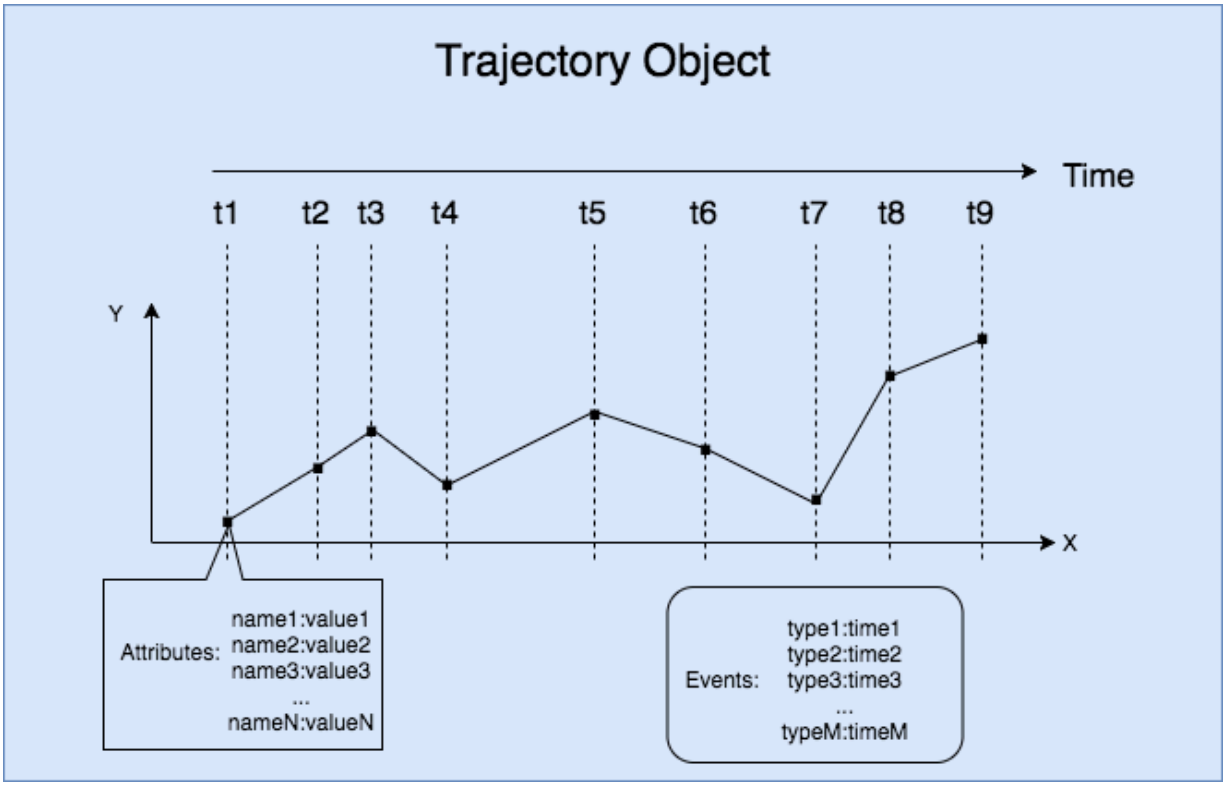
6 Trajectory SQL参考

6.1 基本概念

本章节将为您介绍Trajectory SQL的基本概念。

名称	描述
Trajectory Point	轨迹点，是由移动对象在某个时刻所在的空间位置与附带的属性值组成的时空对象，其中空间位置支持二维坐标或三维坐标，属性支持多字段多类型。
Trajectory Object	轨迹对象，是由一系列轨迹点、轨迹事件组成的含时间、空间、属性、事件的高维对象。
Trajectory Timeline	轨迹时间序列：轨迹在时间上连续推进的时间值序列。
Trajectory Spatial	轨迹空间对象，轨迹在空间上的Geometry对象，通常为linestring。
Trajectory Leaf	轨迹叶子，这里指轨迹点Point，即移动对象在某个时刻的空间位置。
Trajectory Attributes	轨迹属性信息（以下简称属性），移动对象在不同轨迹点上所具有的属性信息，比如速度信息、方向信息等。
Trajectory Attribute Field	轨迹属性字段（以下简称字段），轨迹属性中的某个字段，比如速度字段，轨迹属性字段值的个数与轨迹点个数一致。
Trajectory Field Value	轨迹属性值，轨迹属性在某一时刻某个字段的值。
Trajectory Events	轨迹事件，在轨迹行程中发生的额外事件，比如汽车行程轨迹中的加油事件、抛锚事件、锁车事件等，由事件类型ID和事件时间组成。

轨迹示意图



6.2 构造函数

6.2.1 构造函数概述

构造函数包括由JSON或数组构造轨迹对象的函数及轨迹追加函数。

6.2.2 ST_makeTrajectory

构造一个trajectory对象。

语法

```
trajectory  ST_makeTrajectory (leaftype type, geometry spatial,
tsrange timespan , cstring attrs_json);
trajectory  ST_makeTrajectory (leaftype type, geometry spatial,
timestamp start,  timestamp end ,  cstring attrs_json);
trajectory  ST_makeTrajectory (leaftype type, geometry spatial,
timestamp[]  timeline, cstringattrs_json );
trajectory  ST_makeTrajectory (leaftype type, float8[] x, float8[]
y, integer srid, timestamp[] timeline, text[] attr_field_names, int4
[] attr_int4, float8[] attr_float8, text[] attr_cstring , anyarrayat
tr_any );
```

参数

参数名称	描述
type	轨迹的类型，目前只支持 ST_POINT。

参数名称	描述
spatial	基于 LineString/Point类型的轨迹空间对象。
timespan	包含开始时间和结束时间的tsrange。
start	轨迹的开始时间。
end	轨迹的结束时间。
timeline	轨迹的时间序列，数量必须和linestring的点数量一致。
attrs_json	轨迹属性和事件，JSON格式，可以为空值。
attr_field_names	表示轨迹属性的所有字段名称（数组）。
x	用于构建几何对象的x坐标（数组）。
y	用于构建几何对象的y坐标（数组）。
srid	轨迹空间参考。必须存在。

1. attr_json的格式为：

```
{
  "leafcount": 3,
  "attributes": {
    "velocity": {
      "type": "integer",
      "length": 2,
      "nullable": true,
      "value": [
        120,
        null,
        140
      ]
    },
    "accuracy": {
      "type": "float",
      "length": 4,
      "nullable": false,
      "value": [
        120,
        130,
        140
      ]
    },
    "bearing": {
      "type": "float",
      "length": 8,
      "nullable": false,
      "value": [
        120,
        130,
        140
      ]
    },
    "vesname": {
      "type": "string",
      "length": 20,
```

```
      "nullable": true,
      "value": [
        "dsff",
        "fgsd",
        null
      ]
    },
    "active": {
      "type": "timestamp",
      "nullable": false,
      "value": [
        "Fri Jan 01 14:30:00 2010",
        "Fri Jan 01 15:00:00 2010",
        "Fri Jan 01 15:30:00 2010"
      ]
    }
  },
  "events": [
    {
      "1": "Fri Jan 01 14:30:00 2010"
    },
    {
      "2": "Fri Jan 01 15:00:00 2010"
    },
    {
      "3": "Fri Jan 01 15:30:00 2010"
    }
  ]
}
```

leafcount为轨迹包含的轨迹点个数，必须与spatial对象中包含的空间点个数一致，同时也是每个属性字段包含的元素值个数，所有属性元素个数都必须一致；

attributes为属性项，包含所有属性的字段定义及值序列，与leafcount同时存在，属性定义及要求：

- 属性名称最多60个字符；
- type为字段类型，支持integer，float，string, timestamp，bool五种数据类型；
- length为字段长度，integer支持长度为1、2、4、8; float支持长度为4、8；string可自定义长度，不指定时默认长度为64，最大长度为253，该长度值为字符实际个数，不包含末尾的结束标识；timestamp长度可不指定，默认为8；bool长度可不指定，默认为1；
- nullable为字段是否允许为空，true为允许为空，false不允许为空，默认值为true；
- value为字段值序列，用json数组表达，单个元素值为空用null表达。

events为轨迹事件，用json数组表达多个事件，数组元素用json的“key:value”表达，key为事件类型，value为事件时间。

2. 如果传入的时间参数为 timespan 或者 start、end，则会根据spatial中点的个数进行插值。

3. 如果形式4不满足实际使用，可以自定义MakeTrajectory函数，前六个为固定参数，后面的参数可以根据实际情况进行定制：

```
CREATE OR REPLACE FUNCTION  _ST_MakeTrajectory(type leaftype, x
float8[], y float8[] , srid integer, timespan timestamp[],
    attrs_name cstring[], attr1 float8[], attr2 float4[], attr3
timestamp[])
    RETURNS trajectory
    AS '$libdir/libpg-trajectoryxx','sqltr_traj_make_all_array'
    LANGUAGE 'c' IMMUTABLE Parallel SAFE;
```

示例

```
-- (1) ST_MakeTrajectory with timestamp range
select ST_MakeTrajectory('STPOINT'::leaftype, st_geomfromtext('
LINESTRING (114 35, 115 36, 116 37)', 4326), '[2010-01-01 14:30, 2010
-01-01 15:30)'::tsrange, '{"leafcount":3,"attributes":{"velocity": {"
type": "integer", "length": 2,"nullable" : true,"value": [120, 130,
140]}, "accuracy": {"type": "float", "length": 4, "nullable" : false
,"value": [120, 130, 140]}, "bearing": {"type": "float", "length":
8, "nullable" : false,"value": [120, 130, 140]}, "vesname": {"type":
 "string", "length": 20, "nullable" : true,"value": ["adsf", "sdf",
 "sdfff"]}, "active": {"type": "timestamp", "nullable" : false,"value
": ["Fri Jan 01 14:30:00 2010", "Fri Jan 01 15:00:00 2010", "Fri Jan
 01 15:30:00 2010"]}}, "events": [{"1" : "Fri Jan 01 14:30:00 2010"},
 {"2" : "Fri Jan 01 15:00:00 2010"}, {"3" : "Fri Jan 01 15:30:00 2010
"}]}');                          st_maketrajectory
-----------------------------------------------------------------------------
 {"trajectory":{"version":1,"type":"STPOINT","leafcount":3,"start_time
":"2010-01-01 14:30:00","end_time":"2010-01-01 15:30:00","spatial":"
SRID=4326;LINESTRING(114 35,115 36,116 37)","timeline":["2010-01-01
 14:30:00","2010-01-01 15:00:00","2010-01-01 15:30:00"],"attributes
":{"leafcount":3,"velocity":{"type":"integer","length":2,"nullable":
true,"value":[120,130,140]},"accuracy":{"type":"float","length":4,"
nullable":false,"value":[120.0,130.0,140.0]},"bearing":{"type":"float
","length":8,"nullable":false,"value":[120.0,130.0,140.0]},"vesname":
{"type":"string","length":20,"nullable":true,"value":["adsf","sdf","
sdfff"]},"active":{"type":"timestamp","length":8,"nullable":false,"
value":["2010-01-01 14:30:00","2010-01-01 15:00:00","2010-01-01 15:30:
00"]}},"events":[{"1":"2010-01-01 14:30:00"},{"2":"2010-01-01 15:00:00
"},{"3":"2010-01-01 15:30:00"}]}}
(1 row)

-- (2) ST_MakeTrajectory with start timestamp and end timestamp
select ST_MakeTrajectory('STPOINT'::leaftype, st_geomfromtext('
LINESTRING (114 35, 115 36, 116 37)', 4326), '2010-01-01 14:30'::
timestamp, '2010-01-01 15:30'::timestamp, '{"leafcount":3,"attributes
":{"velocity": {"type": "integer", "length": 2,"nullable" : true,"
value": [120, 130, 140]}, "accuracy": {"type": "float", "length": 4,
 "nullable" : false,"value": [120, 130, 140]}, "bearing": {"type": "
float", "length": 8, "nullable" : false,"value": [120, 130, 140]}, "
vesname": {"type": "string", "length": 20, "nullable" : true,"value":
 ["adsf", "sdf", "sdfff"]}, "active": {"type": "timestamp", "nullable
" : false,"value": ["Fri Jan 01 14:30:00 2010", "Fri Jan 01 15:00:00
2010", "Fri Jan 01 15:30:00 2010"]}}, "events": [{"1" : "Fri Jan 01 14
:30:00 2010"}, {"2" : "Fri Jan 01 15:00:00 2010"}, {"3" : "Fri Jan 01
15:30:00 2010"}]}');                          st_maketrajectory
-----------------------------------------------------------------------------
 {"trajectory":{"version":1,"type":"STPOINT","leafcount":3,"start_time
":"2010-01-01 14:30:00","end_time":"2010-01-01 15:30:00","spatial":"
SRID=4326;LINESTRING(114 35,115 36,116 37)","timeline":["2010-01-01
```

```
 14:30:00","2010-01-01 15:00:00","2010-01-01 15:30:00"],"attributes
":{"leafcount":3,"velocity":{"type":"integer","length":2,"nullable":
true,"value":[120,130,140]},"accuracy":{"type":"float","length":4,"
nullable":false,"value":[120.0,130.0,140.0]},"bearing":{"type":"float
","length":8,"nullable":false,"value":[120.0,130.0,140.0]},"vesname":
{"type":"string","length":20,"nullable":true,"value":["adsf","sdf","
sdfff"]},"active":{"type":"timestamp","length":8,"nullable":false,"
value":["2010-01-01 14:30:00","2010-01-01 15:00:00","2010-01-01 15:30:
00"]}},"events":[{"1":"2010-01-01 14:30:00"},{"2":"2010-01-01 15:00:00
"},{"3":"2010-01-01 15:30:00"}]}}
(1 row)

-- (3) ST_MakeTrajectory with timestamp array
select ST_MakeTrajectory('STPOINT'::leaftype, st_geomfromtext('
LINESTRING (114 35, 115 36, 116 37)', 4326), ARRAY['2010-01-01 14
:30'::timestamp, '2010-01-01 15:00'::timestamp, '2010-01-01 15:30
'::timestamp], '{"leafcount":3,"attributes":{"velocity": {"type": "
integer", "length": 2,"nullable" : true,"value": [120, 130, 140]}, "
accuracy": {"type": "float", "length": 4, "nullable" : false,"value":
 [120, 130, 140]}, "bearing": {"type": "float", "length": 8, "nullable
" : false,"value": [120, 130, 140]}, "vesname": {"type": "string", "
length": 20, "nullable" : true,"value": ["adsf", "sdf", "sdfff"]}, "
active": {"type": "timestamp", "nullable" : false,"value": ["Fri Jan
 01 14:30:00 2010", "Fri Jan 01 15:00:00 2010", "Fri Jan 01 15:30:00
2010"]}}, "events": [{"1" : "Fri Jan 01 14:30:00 2010"}, {"2" : "Fri
Jan 01 15:00:00 2010"}, {"3" : "Fri Jan 01 15:30:00 2010"}]}');
                 st_maketrajectory
-----------------------------------------------------------------------------
 {"trajectory":{"version":1,"type":"STPOINT","leafcount":3,"start_time
":"2010-01-01 14:30:00","end_time":"2010-01-01 15:30:00","spatial":"
SRID=4326;LINESTRING(114 35,115 36,116 37)","timeline":["2010-01-01
 14:30:00","2010-01-01 15:00:00","2010-01-01 15:30:00"],"attributes
":{"leafcount":3,"velocity":{"type":"integer","length":2,"nullable":
true,"value":[120,130,140]},"accuracy":{"type":"float","length":4,"
nullable":false,"value":[120.0,130.0,140.0]},"bearing":{"type":"float
","length":8,"nullable":false,"value":[120.0,130.0,140.0]},"vesname":
{"type":"string","length":20,"nullable":true,"value":["adsf","sdf","
sdfff"]},"active":{"type":"timestamp","length":8,"nullable":false,"
value":["2010-01-01 14:30:00","2010-01-01 15:00:00","2010-01-01 15:30:
00"]}},"events":[{"1":"2010-01-01 14:30:00"},{"2":"2010-01-01 15:00:00
"},{"3":"2010-01-01 15:30:00"}]}}
(1 row)

-- (4) json is null
select ST_MakeTrajectory('STPOINT'::leaftype, st_geomfromtext('
LINESTRING (114 35, 115 36, 116 37)', 4326), '[2010-01-01 14:30, 2010-
01-01 15:30)'::tsrange, null);
                        st_maketrajectory


-----------------------------------------------------------------------------
{"trajectory":{"leafsize":3,"starttime":"Fri Jan 01 14:30:00 2010","
endtime":"Fri Jan 01 15:30:00 2010","spatial":"LINESTRING(114 35,115
36,116 37)","timeline":["Fri Jan 01 14:30:00 2010","Fri Jan 01 15:00:
00 2010","Fri Jan 01 15:30:00 2010"]}}
(1 row)

-- (5) ST_MakeTrajectory make from points
select st_makeTrajectory('STPOINT'::leaftype, ARRAY[1::float8],
ARRAY[2::float8], 4326, ARRAY['2010-01-01 11:30'::timestamp], ARRAY['
velocity'], ARRAY[1::int4], NULL, NULL, NULL::anyarray);
                                                          st_maketra
jectory
```

```
----------------------------------------------------------------------------------
{"trajectory":{"version":1,"type":"STPOINT","leafcount":1,"start_time
":"2010-01-01 11:30:00","end_time":"2010-01-01 11:30:00","spatial":"
SRID=4326;POINT(1 2)","timeline":["2010-01-01 11:30:00"],"attributes":
{"leafcount":1,"velocity":{"type":"integer","length":4,"nullable":true
,"value":[1]}}}}
(1 row)
```

6.2.3 ST_append

向轨迹中追加轨迹点或子轨迹。

语法

```
trajectory ST_append(trajectory traj, geometry spatial, timestamp[]
timespan, text str_attrs_json) ;
trajectory ST_append(trajectory traj, trajectroy tail) ;
```

参数

参数名称	描述
traj	原轨迹。
spatial	追加的轨迹空间对象。
timespan	追加的轨迹的时间数组，可为时间序列。
str_attrs_json	追加轨迹的属性信息，参见 ST_MakeTrajectory。
tail	追加的轨迹。

示例

```
With traj AS (  Select ST_makeTrajectory('STPOINT', 'LINESTRING(1 1, 6
 6, 9 8)'::geometry, '[2010-01-01 11:30, 2010-01-01 15:00)'::tsrange,
 '{"leafcount":3,"attributes":{"velocity": {"type": "integer", "length
": 2,"nullable" : true,"value": [120, 130, 140]}, "accuracy": {"type":
 "float", "length": 4, "nullable" : false,"value": [120, 130, 140]}, "
bearing": {"type": "float", "length": 8, "nullable" : false,"value":
 [120, 130, 140]}, "acceleration": {"type": "string", "length": 20, "
nullable" : true,"value": ["120", "130", "140"]}, "active": {"type":
 "timestamp", "nullable" : false,"value": ["Fri Jan 01 14:30:00 2010
", "Fri Jan 01 15:00:00 2010", "Fri Jan 01 15:30:00 2010"]}}, "events
": [{"1" : "Fri Jan 01 14:30:00 2010"}, {"2" : "Fri Jan 01 15:00:00
2010"}, {"3" : "Fri Jan 01 15:30:00 2010"}]}') a,  ST_makeTrajectory('
STPOINT', 'LINESTRING(7 7, 3 4, 1 5)'::geometry, '[2010-01-02 15:30,
2010-01-02 18:00)'::tsrange,  '{"leafcount":3,"attributes":{"velocity
": {"type": "integer", "length": 2,"nullable" : true,"value": [121,
131, 141]}, "accuracy": {"type": "float", "length": 4, "nullable" :
false,"value": [121, 131, 141]}, "bearing": {"type": "float", "length
": 8, "nullable" : false,"value": [121, 131, 141]}, "acceleration": {"
type": "string", "length": 20, "nullable" : true,"value": ["121", "131
", "141"]}, "active": {"type": "timestamp", "nullable" : false,"value
": ["Fri Jan 02 14:30:00 2010", "Fri Jan 02 15:00:00 2010", "Fri Jan
 02 15:30:00 2010"]}}, "events": [{"1" : "Fri Jan 02 14:30:00 2010"},
 {"2" : "Fri Jan 02 15:00:00 2010"}, {"3" : "Fri Jan 02 15:30:00 2010
"}]}') b)Select ST_Append(a, b) from traj;
```

```
                  st_append


-------------------------------------------------------------------------------
 {"trajectory":{"version":1,"type":"STPOINT","leafcount":6,"start_time
":"2010-01-01 11:30:00","end_time":"2010-01-02 18:00:00","spatial":"
LINESTRING(1 1,6 6,9 8,7 7,3 4,1 5)","timeline":["2010-01-01 11:30:00
","2010-01-01 13:15:00","2010-01-01 15:00:00","2010-01-02 15:30:00","
2010-01-02 16:45:00","2010-01-02 18:00:00"],"attributes":{"leafcount":
6,"velocity":{"type":"integer","length":2,"nullable":true,"value":[120
,130,140,121,131,141]},"accuracy":{"type":"float","length":4,"nullable
":false,"value":[120.0,130.0,140.0,121.0,131.0,141.0]},"bearing":{"
type":"float","length":8,"nullable":false,"value":[120.0,130.0,140.
0,121.0,131.0,141.0]},"acceleration":{"type":"string","length":20,"
nullable":true,"value":["120","130","140","121","131","141"]},"active
":{"type":"timestamp","length":8,"nullable":false,"value":["2010-01-01
 14:30:00","2010-01-01 15:00:00","2010-01-01 15:30:00","2010-01-02 14:
30:00","2010-01-02 15:00:00","2010-01-02 15:30:00"]}},"events":[{"1":"
2010-01-01 14:30:00"},{"2":"2010-01-01 15:00:00"},{"3":"2010-01-01 15:
30:00"},{"1":"2010-01-02 14:30:00"},{"2":"2010-01-02 15:00:00"},{"3":"
2010-01-02 15:30:00"}]}}
(1 row)
```

6.3 编辑与处理函数

6.3.1 ST_Compress

将trajectory对象按一定规则进行压缩。

语法

```
trajectory ST_Compress (trajectory traj, float8 dist);
trajectory ST_Compress (trajectory traj, float8 dist, float8 angle,
float8  acceleration);
trajectory ST_Compress (trajectory traj, float8 dist, float8 angle,
float8  acceleration, cstring velocity_field);
```

参数

参数名称	描述
traj	**原始轨迹。**
dist	**欧式距离偏移阈值，指定后可以保留轨迹空间上的整体趋势。**
angle	**角度偏移阈值，指定后可以保留方向变化较大的轨迹点。**
acceleration	**加速度阈值，指定后可以保留速度变化较大的轨迹点。**
velocity_field	**轨迹中的速度属性名称，指定后用该属性计算加速度。**

描述

- 根据指定的阈值对轨迹进行有损压缩，返回压缩后的轨迹对象。
  - 形式一：指定空间距离偏移值进行压缩，只能保留轨迹空间的整体趋势，对于折返点、加速度点等带有重要信息的轨迹点有可能会被删除。
  - 形式二：可以同时指定空间距离偏移阈值、角度偏移阈值、加速度阈值三个参数对轨迹进行压缩，在保留轨迹空间整体趋势的同时，可以保留方向变化较大及速度变化较大的重要轨迹点。也可以指定三个参数中的任何一个或者二个阈值进行压缩。
  - 形式三：功能等同于形式二，用于轨迹属性中含有速度字段的情况，指定速度字段名称后，直接根据速度属性值计算轨迹点的加速度。
- 对于大轨迹对象的压缩，本函数支持GPU加速计算，如果运行环境中有GPU设备，会自动启动GPU加速。

示例

```
--创建数据
Create table If not exists traj_test(id integer, mmsi integer,traj
trajectory);
INSERT INTO traj_test(mmsi, traj) VALUES(477027500,ST_makeTrajectory
('STPOINT'::leaftype, 'LINESTRING(-179.48077 51.72814,-179.47416 51.
73714,-179.47187 51.74027,-179.46964 51.74325,-179.46731 51.74634,-179
.46502 51.74934,-179.46183 51.75378,-179.45943 51.75736,-179.45560 51.
76273,-179.44845 51.77186,-179.43419 51.78977,-179.42595 51.80094,-179
.42343 51.80411,-179.42078 51.80719,-179.41821 51.81025,-179.41562 51.
81308,-179.41259 51.81643,-179.41001 51.81941,-179.40751 51.82223,-179
.40497 51.82505,-179.40242 51.82796,-179.39981 51.83095,-179.39734 51.
83398,-179.39499 51.83709,-179.39264 51.84023,-179.39037 51.84333,-179
.38699 51.84791,-179.38467 51.85114,-179.38216 51.85439,-179.37997 51.
85762,-179.37772 51.86144,-179.37474 51.86568,-179.37219 51.86869,-179
.36983 51.87156,-179.36755 51.87467,-179.36001 51.88423,-179.35754 51
.88712,-179.34216 51.90644,-179.33935 51.90995,-179.33704 51.91298,-
179.18826 52.10105,-179.18096 52.11031,-179.17504 52.11786,-179.16482
 52.12996,-179.16233 52.13289,-179.15967 52.13590,-179.14599 52.15132
,-177.76666 52.85042,-177.48459 52.89898,-177.47841 52.90001,-177.
47319 52.90084,-177.46251 52.90268,-177.38188 52.91595,-177.37102 52.
91765,-177.36378 52.91877,-177.34492 52.92173,-177.33217 52.92364,-177
.32581 52.92468,-177.31238 52.92697,-177.03751 52.97394,-176.93063 52
.99160,-176.92406 52.99265,-176.91471 52.99423,-176.90643 52.99554,-
176.89912 52.99674,-176.89246 52.99791,-176.88342 52.99942,-176.87697
 53.00060,-176.86594 53.00256,-176.85946 53.00370,-176.85294 53.00481
,-176.84640 53.00592,-176.83985 53.00705,-176.83238 53.00830,-176.
82589 53.00950,-176.81848 53.01084,-176.80553 53.01310,-176.79879 53.
01419,-176.79115 53.01548,-176.78466 53.01668,-176.77901 53.01765,-176
.77256 53.01879,-176.76301 53.02039,-176.75649 53.02141,-176.74700 53
.02296,-176.73757 53.02450,-176.71683 53.02795,-176.70741 53.02950,-
176.68481 53.03327)'::geometry, ARRAY['2017-01-15 09:06:39'::timestamp
,'2017-01-15 09:10:08'::timestamp,'2017-01-15 09:11:20'::timestamp
,'2017-01-15 09:12:29'::timestamp,'2017-01-15 09:13:39'::timestamp
,'2017-01-15 09:14:48'::timestamp,'2017-01-15 09:16:28'::timestamp
,'2017-01-15 09:17:48'::timestamp,'2017-01-15 09:19:48'::timestamp
,'2017-01-15 09:23:19'::timestamp,'2017-01-15 09:30:28'::timestamp
,'2017-01-15 09:34:40'::timestamp,'2017-01-15 09:35:49'::timestamp
,'2017-01-15 09:36:59'::timestamp,'2017-01-15 09:38:09'::timestamp
,'2017-01-15 09:39:18'::timestamp,'2017-01-15 09:40:40'::timestamp
```

```
,'2017-01-15 09:41:49'::timestamp,'2017-01-15 09:42:58'::timestamp
,'2017-01-15 09:44:08'::timestamp,'2017-01-15 09:45:18'::timestamp
,'2017-01-15 09:46:29'::timestamp,'2017-01-15 09:47:38'::timestamp
,'2017-01-15 09:48:49'::timestamp,'2017-01-15 09:49:58'::timestamp
,'2017-01-15 09:51:08'::timestamp,'2017-01-15 09:52:49'::timestamp
,'2017-01-15 09:53:58'::timestamp,'2017-01-15 09:55:09'::timestamp
,'2017-01-15 09:56:18'::timestamp,'2017-01-15 09:57:38'::timestamp
,'2017-01-15 09:59:09'::timestamp,'2017-01-15 10:00:20'::timestamp
,'2017-01-15 10:01:29'::timestamp,'2017-01-15 10:02:39'::timestamp
,'2017-01-15 10:06:29'::timestamp,'2017-01-15 10:07:40'::timestamp
,'2017-01-15 10:15:00'::timestamp,'2017-01-15 10:16:20'::timestamp
,'2017-01-15 10:17:29'::timestamp,'2017-01-15 11:30:09'::timestamp
,'2017-01-15 11:33:58'::timestamp,'2017-01-15 11:36:58'::timestamp
,'2017-01-15 11:42:00'::timestamp,'2017-01-15 11:43:10'::timestamp
,'2017-01-15 11:44:20'::timestamp,'2017-01-15 11:50:28'::timestamp
,'2017-01-15 18:01:00'::timestamp,'2017-01-15 18:54:13'::timestamp
,'2017-01-15 18:55:21'::timestamp,'2017-01-15 18:56:22'::timestamp
,'2017-01-15 18:58:21'::timestamp,'2017-01-15 19:13:21'::timestamp
,'2017-01-15 19:15:21'::timestamp,'2017-01-15 19:16:41'::timestamp
,'2017-01-15 19:20:11'::timestamp,'2017-01-15 19:22:31'::timestamp
,'2017-01-15 19:23:41'::timestamp,'2017-01-15 19:26:10'::timestamp
,'2017-01-15 20:15:49'::timestamp,'2017-01-15 20:34:39'::timestamp
,'2017-01-15 20:35:49'::timestamp,'2017-01-15 20:37:30'::timestamp
,'2017-01-15 20:39:00'::timestamp,'2017-01-15 20:40:19'::timestamp,'
2017-01-15 20:41:30'::timestamp,'2017-01-15 20:43:08'::timestamp,'2017
-01-15 20:44:19'::timestamp,'2017-01-15 20:46:19'::timestamp,'2017-
01-15 20:47:29'::timestamp,'2017-01-15 20:48:40'::timestamp,'2017-01
-15 20:49:49'::timestamp,'2017-01-15 20:50:59'::timestamp,'2017-01-
15 20:52:21'::timestamp,'2017-01-15 20:53:29'::timestamp,'2017-01-15
 20:54:50'::timestamp,'2017-01-15 20:57:09'::timestamp,'2017-01-15
20:58:20'::timestamp,'2017-01-15 20:59:40'::timestamp,'2017-01-15 21
:00:49'::timestamp,'2017-01-15 21:01:50'::timestamp,'2017-01-15 21:
02:58'::timestamp,'2017-01-15 21:04:40'::timestamp,'2017-01-15 21:05
:50'::timestamp,'2017-01-15 21:07:29'::timestamp,'2017-01-15 21:09:
11'::timestamp,'2017-01-15 21:12:49'::timestamp,'2017-01-15 21:14:30
'::timestamp,'2017-01-15 21:18:30'::timestamp], '{"leafcount": 89,"
attributes" : {"sog" : {"type":"float","length":8,"nullable":false,"
value":[10.5,10.4,10.5,10.7,10.8,10.3,10.7,10.4,10.5,10.1,10.2,11.0,11
.2,10.8,10.3,10.3,10.1,10.7,10.6,10.0,10.3,10.5,10.6,10.3,10.8,10.9,10
.8,10.8,10.8,11.0,11.2,11.2,10.3,10.2,10.8,10.0,10.4,10.7,10.2,10.6,9
.1,10.2,10.1,9.7,10.4,10.6,9.9,12.3,12.1,12.0,12.0,12.2,12.3,12.2,12.3
,12.2,12.3,12.2,12.2,12.8,12.8,12.9,12.5,12.6,12.5,12.6,12.6,12.3,12.6
,12.6,12.5,12.7,12.8,12.5,12.7,12.5,12.8,13.0,12.9,12.6,12.9,12.8,12.7
,12.8,13.0,12.7,12.8,12.6,12.7]}, "cog" : {"type":"float","length":8,"
nullable":false,"value":[23.3,25.7,25.9,23.6,25.3,24.1,23.0,21.6,20.7,
24.8,22.4,28.5,23.1,30.3,26.2,28.1,25.1,28.7,31.4,28.2,30.4,29.4,29.2,
23.0,25.1,25.1,23.5,22.7,27.1,23.3,19.2,27.1,31.0,28.8,22.0,30.1,24.6,
26.2,26.7,24.7,26.8,29.5,19.9,30.1,28.8,28.7,30.0,74.2,69.1,75.2,81.3,
81.3,80.1,72.6,82.4,74.2,74.9,67.7,73.4,74.2,72.2,80.5,78.6,77.3,70.9,
80.1,85.4,71.9,67.0,77.5,77.5,72.2,70.5,72.6,70.8,77.8,71.2,71.2,73.8,
75.4,67.1,77.5,74.3,76.9,80.1,72.8,76.0,75.4,72.9]},"heading" : {"type
":"float","length":8,"nullable":false,"value":[22.0,23.0,23.0,23.0,23.
0,21.0,21.0,25.0,24.0,26.0,25.0,27.0,28.0,29.0,31.0,30.0,28.0,29.0,29.
0,28.0,28.0,27.0,24.0,24.0,25.0,25.0,25.0,26.0,25.0,24.0,25.0,29.0,31.
0,28.0,29.0,31.0,28.0,29.0,29.0,29.0,27.0,27.0,26.0,26.0,26.0,27.0,27.
0,69.0,71.0,72.0,72.0,71.0,73.0,72.0,72.0,72.0,72.0,71.0,71.0,72.0,72.
0,72.0,73.0,72.0,71.0,72.0,72.0,72.0,71.0,72.0,72.0,73.0,71.0,72.0,71.
0,72.0,73.0,72.0,72.0,72.0,72.0,73.0,74.0,73.0,73.0,73.0,73.0,73.0,73.
0]}}}'));

--轨迹压缩
select st_compress(traj,0.001) as traj from traj_test;
           traj
------------------------------------------------------------
```

```
 {"trajectory":{"version":1,"type":"STPOINT","leafcount":8,"start_time
":"2017-01-15 09:06:39","end_time":"2017-01-15 21:18:30","spatial":"
LINESTRING(-179.48077 51.72814,-179.42595 51.80094,-179.39734 51.83398
,-179.37474 51.86568,-179.17504 52.11786,-179.14599 52.15132,-177.
76666 52.85042,-176.68481 53.03327)","timeline":["2017-01-15 09:06:39
","2017-01-15 09:34:40","2017-01-15 09:47:38","2017-01-15 09:59:09","
2017-01-15 11:36:58","2017-01-15 11:50:28","2017-01-15 18:01:00","2017
-01-15 21:18:30"],"attributes":{"leafcount":8,"sog":{"type":"float","
length":8,"nullable":false,"value":[10.5,11.0,10.6,11.2,10.1,9.9,12.
3,12.7]},"cog":{"type":"float","length":8,"nullable":false,"value":[
23.3,28.5,29.2,27.1,19.9,30.0,74.2,72.9]},"heading":{"type":"float","
length":8,"nullable":false,"value":[22.0,27.0,24.0,29.0,26.0,27.0,69.0
,73.0]}}}}
(1 row)

select st_compress(traj,0.001,null,null) as traj from traj_test where
traj_id=5;
           traj
-----------------------------------------------------------------
{"trajectory":{"type":"STPOINT","leafsize":8,"starttime":"2017-01-15
09:06:39","endtime":"2017-01-15 21:18:30","spatial":"LINESTRING(-179.
48077 51.72814,-179.42595 51.80094,-179.39734 51.83398,-179.37474 51.
86568,-179.17504 52.11786,-179.14599 52.15132,-177.76666 52.85042,-176
.68481 53.03327)","timeline":["2017-01-15 09:06:39","2017-01-15 09:34
:40","2017-01-15 09:47:38","2017-01-15 09:59:09","2017-01-15 11:36:58
","2017-01-15 11:50:28","2017-01-15 18:01:00","2017-01-15 21:18:30"],"
themeline":{"leafs":8,"sog":[10.5,11.0,10.6,11.2,10.1,9.9,12.3,12.7],"
cog":[23.3,28.5,29.2,27.1,19.9,30.0,74.2,72.9],"heading":[22.0,27.0,24
.0,29.0,26.0,27.0,69.0,73.0]}}}
(1 row)

select st_compress(traj,0.001,5,0.3) as traj from traj_test;
           traj
-----------------------------------------------------------------
{"trajectory":{"version":1,"type":"STPOINT","leafcount":13,"start_time
":"2017-01-15 09:06:39","end_time":"2017-01-15 21:18:30","spatial":"
LINESTRING(-179.48077 51.72814,-179.42595 51.80094,-179.39734 51.83398
,-179.37474 51.86568,-179.35754 51.88712,-179.18826 52.10105,-179.
17504 52.11786,-179.14599 52.15132,-177.76666 52.85042,-177.47841 52
.90001,-177.47319 52.90084,-176.83238 53.0083,-176.68481 53.03327)","
timeline":["2017-01-15 09:06:39","2017-01-15 09:34:40","2017-01-15
09:47:38","2017-01-15 09:59:09","2017-01-15 10:07:40","2017-01-15 11
:30:09","2017-01-15 11:36:58","2017-01-15 11:50:28","2017-01-15 18:01
:00","2017-01-15 18:55:21","2017-01-15 18:56:22","2017-01-15 20:52:21
","2017-01-15 21:18:30"],"attributes":{"leafcount":13,"sog":{"type":"
float","length":8,"nullable":false,"value":[10.5,11.0,10.6,11.2,10.4,
9.1,10.1,9.9,12.3,12.0,12.0,12.5,12.7]},"cog":{"type":"float","length
":8,"nullable":false,"value":[23.3,28.5,29.2,27.1,24.6,26.8,19.9,30.
0,74.2,75.2,81.3,72.6,72.9]},"heading":{"type":"float","length":8,"
nullable":false,"value":[22.0,27.0,24.0,29.0,28.0,27.0,26.0,27.0,69.0,
72.0,72.0,72.0,73.0]}}}}
(1 row)

select st_compress(traj,0.001,5,1.1,'sog') as traj from traj_test;

            traj
-----------------------------------------------------------------
{"trajectory":{"version":1,"type":"STPOINT","leafcount":10,"start_time
":"2017-01-15 09:06:39","end_time":"2017-01-15 21:18:30","spatial":"
LINESTRING(-179.48077 51.72814,-179.42595 51.80094,-179.39734 51.83398
,-179.37474 51.86568,-179.33704 51.91298,-179.18826 52.10105,-179.
17504 52.11786,-179.14599 52.15132,-177.76666 52.85042,-176.68481 53.
03327)","timeline":["2017-01-15 09:06:39","2017-01-15 09:34:40","2017
-01-15 09:47:38","2017-01-15 09:59:09","2017-01-15 10:17:29","2017-01
```

```
-15 11:30:09","2017-01-15 11:36:58","2017-01-15 11:50:28","2017-01-15
 18:01:00","2017-01-15 21:18:30"],"attributes":{"leafcount":10,"sog":
{"type":"float","length":8,"nullable":false,"value":[10.5,11.0,10.6,
11.2,10.6,9.1,10.1,9.9,12.3,12.7]},"cog":{"type":"float","length":8,"
nullable":false,"value":[23.3,28.5,29.2,27.1,24.7,26.8,19.9,30.0,74.2,
72.9]},"heading":{"type":"float","length":8,"nullable":false,"value":[
22.0,27.0,24.0,29.0,29.0,27.0,26.0,27.0,69.0,73.0]}}}}
(1 row)
```

6.3.2 ST_CompressSED

将trajectory对象按一定规则进行压缩。

语法

```
trajectory ST_CompressSed (trajectory traj, float8 dist);
```

参数

参数名称	描述
traj	原始轨迹。
dist	时间同步距离(SED)偏移阈值，指定后可以保留轨迹空间上的整体趋势。

描述

计算轨迹点的时间同步距离(SED)，用距离值与阈值比较对轨迹进行有损压缩，返回压缩后的轨迹对象。

示例

```
select st_compressSED(traj, 0.001) as traj from traj_test;
           traj
------------------------------------------------------------
{"trajectory":{"version":1,"type":"STPOINT","leafcount":12,"start_time
":"2017-01-15 09:06:39","end_time":"2017-01-15 21:18:30","spatial":"
LINESTRING(-179.48077 51.72814,-179.42595 51.80094,-179.39734 51.83398
,-179.37474 51.86568,-179.18826 52.10105,-179.16482 52.12996,-179.
14599 52.15132,-177.76666 52.85042,-177.47319 52.90084,-177.31238 52
.92697,-177.03751 52.97394,-176.68481 53.03327)","timeline":["2017-
01-15 09:06:39","2017-01-15 09:34:40","2017-01-15 09:47:38","2017-01
-15 09:59:09","2017-01-15 11:30:09","2017-01-15 11:42:00","2017-01-15
 11:50:28","2017-01-15 18:01:00","2017-01-15 18:56:22","2017-01-15 19
:26:10","2017-01-15 20:15:49","2017-01-15 21:18:30"],"attributes":{"
leafcount":12,"sog":{"type":"float","length":8,"nullable":false,"value
":[10.5,11.0,10.6,11.2,9.1,9.7,9.9,12.3,12.0,12.2,12.8,12.7]},"cog":{"
type":"float","length":8,"nullable":false,"value":[23.3,28.5,29.2,27.1
,26.8,30.1,30.0,74.2,81.3,73.4,74.2,72.9]},"heading":{"type":"float","
length":8,"nullable":false,"value":[22.0,27.0,24.0,29.0,27.0,26.0,27.0
,69.0,72.0,71.0,72.0,73.0]}}}}
```

```
(1 row)
```

6.3.3 ST_attrDeduplicate

指定轨迹属性字段名称，抽除该属性字段中值重复的轨迹点，轨迹首尾点必须保留，返回抽除后的轨迹。

语法

```
trajectory ST_attrDeduplicate(trajectory traj, cstring attr_field_name
);
```

参数

参数名称	描述
traj	**轨迹对象。**
attr_field_name	**指定的属性名称。**

示例

```
select st_attrDeduplicate(ST_makeTrajectory('STPOINT'::leaftype, '
LINESTRING(-179.48077 51.72814,-179.46731 51.74634,-179.46502 51.74934
,-179.46183 51.75378,-179.45943 51.75736,-179.45560 51.76273,-179.
44845 51.77186,-179.43419 51.78977,-179.41259 51.81643,-179.41001 51.
81941,-179.40751 51.82223,-179.40497 51.82505,-179.40242 51.82796,-179
.39981 51.83095,-179.39734 51.83398,-179.39499 51.83709)'::geometry,
ARRAY['2017-01-15 09:06:39'::timestamp,'2017-01-15 09:13:39','2017-01
-15 09:14:48','2017-01-15 09:16:28','2017-01-15 09:17:48','2017-01-15
 09:19:48','2017-01-15 09:23:19','2017-01-15 09:30:28','2017-01-15 09
:34:40','2017-01-15 09:36:59','2017-01-15 09:38:09','2017-01-15 09:39
:18','2017-01-15 09:40:40','2017-01-15 09:47:38','2017-01-15 09:48:49
','2017-01-15 21:18:30'], '{"leafcount": 16, "attributes" : {"heading
" : {"type": "float", "length": 4, "nullable" : false,"value": [23.0,
23.0,23.0,23.0,21.0,21.0,72.0,72.0,72.0,72.0,73.0,74.0,73.0,73.0,73.0,
73.0]}}}' ),'heading') as traj;


                                                        traj


-------------------------------------------------------------------------------
 {"trajectory":{"version":1,"type":"STPOINT","leafcount":6,"start_time
":"2017-01-15 09:17:48","end_time":"2017-01-15 21:18:30","spatial":"
LINESTRING(-179.45943 51.75736,-179.44845 51.77
186,-179.40751 51.82223,-179.40497 51.82505,-179.39499 51.83709)","
timeline":["2017-01-15 09:17:48","2017-01-15 09:23:19","2017-01-15 09:
38:09","2017-01-15 09:39:18","2017-01-15 21:18:3
0"],"attributes":{"leafcount":6,"heading":{"type":"float","length":4,"
nullable":false,"value":[23.0,21.0,72.0,73.0,74.0,73.0]}}}}
```

```
(1 row)
```

6.3.4 ST_sort

轨迹按时间序列从小到大重新排序，并返回新的轨迹对象。

语法

```
trajectory ST_sort(trajectory traj);
```

参数

参数名称	描述
traj	**轨迹对象。**

示例

```
select st_sort(ST_makeTrajectory('STPOINT'::leaftype, 'LINESTRING(-179
.48077 51.72814,-179.46731 51.74634,-179.46502 51.74934,-179.46183 51
.75378,-179.45943 51.75736,-179.45560 51.76273,-179.44845 51.77186,-
179.43419 51.78977,-179.41259 51.81643,-179.41001 51.81941,-179.40751
 51.82223,-179.40497 51.82505,-179.40242 51.82796,-179.39981 51.83095
,-179.39734 51.83398,-179.39499 51.83709)'::geometry, ARRAY['2017-01
-15 09:06:39'::timestamp,'2017-01-15 09:14:48'::timestamp,'2017-01-
15 09:13:39'::timestamp,'2017-01-15 09:16:28'::timestamp,'2017-01-15
 09:19:48'::timestamp,'2017-01-15 09:17:48'::timestamp,'2017-01-15
09:23:19'::timestamp,'2017-01-15 09:34:40'::timestamp,'2017-01-15 09
:30:28'::timestamp,'2017-01-15 09:36:59'::timestamp,'2017-01-15 09:38
:09'::timestamp,'2017-01-15 09:39:18'::timestamp,'2017-01-15 09:40:
40'::timestamp,'2017-01-15 09:47:38'::timestamp,'2017-01-15 21:18:30
'::timestamp,'2017-01-15 09:48:49'::timestamp], '{"leafcount": 16, "
attributes" : {"heading" : {"type": "integer", "length": 4, "nullable
" : false,"value":[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]}}}' ));

                                                              st_sort


--------------------------------------------------------------------------------
 {"trajectory":{"version":1,"type":"STPOINT","leafcount":16,"
start_time":"2017-01-15 09:06:39","end_time":"2017-01-15 21:18:30","
spatial":"LINESTRING(-179.48077 51.72814,-179.46502 51.7
4934,-179.46731 51.74634,-179.46183 51.75378,-179.4556 51.76273,-179.
45943 51.75736,-179.44845 51.77186,-179.41259 51.81643,-179.43419 51.
78977,-179.41001 51.81941,-179.40751 51.82223,-
179.40497 51.82505,-179.40242 51.82796,-179.39981 51.83095,-179.39499
51.83709,-179.39734 51.83398)","timeline":["2017-01-15 09:06:39","2017
-01-15 09:13:39","2017-01-15 09:14:48","2017-
01-15 09:16:28","2017-01-15 09:17:48","2017-01-15 09:19:48","2017-01-
15 09:23:19","2017-01-15 09:30:28","2017-01-15 09:34:40","2017-01-15
09:36:59","2017-01-15 09:38:09","2017-01-15 09:
39:18","2017-01-15 09:40:40","2017-01-15 09:47:38","2017-01-15 09:48
:49","2017-01-15 21:18:30"],"attributes":{"leafcount":16,"heading":{"
type":"integer","length":4,"nullable":false,"val
ue":[0,2,1,3,5,4,6,8,7,9,10,11,12,13,15,14]}}}}
```

```
(1 row)
```

6.3.5 ST_deviation

计算处理后的轨迹与原始轨迹之间的偏差。

语法

```
trajectory ST_deviation(trajectory traj, trajectory after_oper_traj);
```

参数

参数名称	描述
traj	原始轨迹对象。
after_oper_traj	处理后的轨迹对象（比如压缩后）。

示例

```
select st_deviation(traj, st_compress(traj,0.001)) from traj_test;
    st_deviation
---------------------
0.00919177345596219
(1 row)
```

6.4 属性元数据

6.4.1 ST_attrDefinition

获得轨迹属性定义。

语法

```
text ST_attrDefinition(trajectory traj);
```

参数

参数名称	描述
traj	需要获得属性定义轨迹。

示例

```
With traj AS (
  select '{"trajectory":{"version":1,"type":"STPOINT","leafcount":2,"
start_time":"2010-01-01 11:30:00","end_time":"2010-01-01 12:30:00","
spatial":"SRID=4326;LINESTRING(1 1,3 5)","timeline":["2010-01-01 11:30
:00","2010-01-01 12:30:00"],"attributes":{"leafcount":2,"velocity":{"
type":"integer","length":4,"nullable":true,"value":[1,null]},"speed":
{"type":"float","length":8,"nullable":true,"value":[null,1.0]},"angel
":{"type":"string","length":64,"nullable":true,"value":["test",null
```

```
]}, "tngel2":{"type":"timestamp","length":8,"nullable":true,"value":
["2010-01-01 12:30:00",null]},"bearing":{"type":"bool","length":1,"
nullable":true,"value":[null,true]}}}}'::trajectory a)
Select st_attrDefinition(a) from traj;


st_attrdefinition


--------------------------------------------------------------------------------
 {"size":5,"velocity":{"type":"integer","length":4,"nullable":true
},"speed":{"type":"float","length":8,"nullable":true},"angel":{"type
":"string","length":64,"nullable":true},"tngel2":{"type":"timestamp
","length":8,"nullable":true},"bearing":{"type":"bool","length":1,"
nullable":true}}
(1 row)
```

6.4.2 ST_attrSize

获得轨迹的属性数量。

语法

```
integer ST_attrSize(trajectory traj) ;
```

参数

参数名称	描述
traj	**轨迹对象。**

示例

```
With traj AS (
  select '{"trajectory":{"version":1,"type":"STPOINT","leafcount":2,"
start_time":"2010-01-01 11:30:00","end_time":"2010-01-01 12:30:00","
spatial":"SRID=4326;LINESTRING(1 1,3 5)","timeline":["2010-01-01 11:30
:00","2010-01-01 12:30:00"],"attributes":{"leafcount":2,"velocity":{"
type":"integer","length":4,"nullable":true,"value":[1,null]},"speed":
{"type":"float","length":8,"nullable":true,"value":[null,1.0]},"angel
":{"type":"string","length":64,"nullable":true,"value":["test",null
]}, "tngel2":{"type":"timestamp","length":8,"nullable":true,"value":
["2010-01-01 12:30:00",null]},"bearing":{"type":"bool","length":1,"
nullable":true,"value":[null,true]}}}}'::trajectory a)
Select st_attrSize(a) from traj;
 st_attrsize
-------------
           5
(1 row)
```

6.4.3 ST_attrName

获得轨迹的属性名称。

语法

```
text[] ST_attrName(trajectory traj) ;
```

```
text ST_attrName(trajectory traj, integer index) ;
```

参数

参数名称	描述
traj	轨迹对象。
index	属性索引号，从0开始。

示例

```
With traj AS (
  select '{"trajectory":{"version":1,"type":"STPOINT","leafcount":2,"
start_time":"2010-01-01 11:30:00","end_time":"2010-01-01 12:30:00","
spatial":"SRID=4326;LINESTRING(1 1,3 5)","timeline":["2010-01-01 11:30
:00","2010-01-01 12:30:00"],"attributes":{"leafcount":2,"velocity":{"
type":"integer","length":4,"nullable":true,"value":[1,null]},"speed":
{"type":"float","length":8,"nullable":true,"value":[null,1.0]},"angel
":{"type":"string","length":64,"nullable":true,"value":["test",null
]}, "tngel2":{"type":"timestamp","length":8,"nullable":true,"value":
["2010-01-01 12:30:00",null]},"bearing":{"type":"bool","length":1,"
nullable":true,"value":[null,true]}}}}'::trajectory a)
Select st_attrName(a) from traj;
              st_attrname
---------------------------------------
 {velocity,speed,angel,tngel2,bearing}
(1 row)

With traj AS (
  select '{"trajectory":{"version":1,"type":"STPOINT","leafcount":2,"
start_time":"2010-01-01 11:30:00","end_time":"2010-01-01 12:30:00","
spatial":"SRID=4326;LINESTRING(1 1,3 5)","timeline":["2010-01-01 11:30
:00","2010-01-01 12:30:00"],"attributes":{"leafcount":2,"velocity":{"
type":"integer","length":4,"nullable":true,"value":[1,null]},"speed":
{"type":"float","length":8,"nullable":true,"value":[null,1.0]},"angel
":{"type":"string","length":64,"nullable":true,"value":["test",null
]}, "tngel2":{"type":"timestamp","length":8,"nullable":true,"value":
["2010-01-01 12:30:00",null]},"bearing":{"type":"bool","length":1,"
nullable":true,"value":[null,true]}}}}'::trajectory a)
Select st_attrName(a, 0) from traj;
 st_attrname
-------------
 velocity
(1 row)
```

6.4.4 ST_attrType

获得轨迹属性类型。

语法

```
text ST_attrType(trajectory traj, integer index) ;
```

```
text ST_attrType(trajectory traj, text name) ;
```

参数

参数名称	描述
traj	轨迹对象。
index	索引轨迹项。
name	属性名称。

描述

返回描述属性类型的字符串为以下几种之一：

- integer
- float
- string
- timestamp
- bool

示例

```
With traj AS (
  select '{"trajectory":{"version":1,"type":"STPOINT","leafcount":2,"
start_time":"2010-01-01 11:30:00","end_time":"2010-01-01 12:30:00","
spatial":"SRID=4326;LINESTRING(1 1,3 5)","timeline":["2010-01-01 11:30
:00","2010-01-01 12:30:00"],"attributes":{"leafcount":2,"velocity":{"
type":"integer","length":4,"nullable":true,"value":[1,null]},"speed":
{"type":"float","length":8,"nullable":true,"value":[null,1.0]},"angel
":{"type":"string","length":64,"nullable":true,"value":["test",null
]}, "tngel2":{"type":"timestamp","length":8,"nullable":true,"value":
["2010-01-01 12:30:00",null]},"bearing":{"type":"bool","length":1,"
nullable":true,"value":[null,true]}}}}'::trajectory a)
Select st_attrType(a, 0) from traj;
 st_attrtype
-------------
 integer
(1 row)

With traj AS (
  select '{"trajectory":{"version":1,"type":"STPOINT","leafcount":2,"
start_time":"2010-01-01 11:30:00","end_time":"2010-01-01 12:30:00","
spatial":"SRID=4326;LINESTRING(1 1,3 5)","timeline":["2010-01-01 11:30
:00","2010-01-01 12:30:00"],"attributes":{"leafcount":2,"velocity":{"
type":"integer","length":4,"nullable":true,"value":[1,null]},"speed":
{"type":"float","length":8,"nullable":true,"value":[null,1.0]},"angel
":{"type":"string","length":64,"nullable":true,"value":["test",null
]}, "tngel2":{"type":"timestamp","length":8,"nullable":true,"value":
["2010-01-01 12:30:00",null]},"bearing":{"type":"bool","length":1,"
nullable":true,"value":[null,true]}}}}'::trajectory a)
Select st_attrType(a, 'velocity') from traj;
 st_attrtype
-------------
 integer
```

```
(1 row)
```

6.4.5 ST_attrLength

获得轨迹属性定义长度。

语法

```
integer ST_attrLength(trajectory traj, integer index) ;
integer ST_attrLength(trajectory traj, text name) ;
```

参数

参数名称	描述
traj	**轨迹对象。**
index	**索引轨迹项。**
name	**属性名称。**

示例

```
With traj AS (
  select '{"trajectory":{"version":1,"type":"STPOINT","leafcount":2,"
start_time":"2010-01-01 11:30:00","end_time":"2010-01-01 12:30:00","
spatial":"SRID=4326;LINESTRING(1 1,3 5)","timeline":["2010-01-01 11:30
:00","2010-01-01 12:30:00"],"attributes":{"leafcount":2,"velocity":{"
type":"integer","length":4,"nullable":true,"value":[1,null]},"speed":
{"type":"float","length":8,"nullable":true,"value":[null,1.0]},"angel
":{"type":"string","length":64,"nullable":true,"value":["test",null
]}, "tngel2":{"type":"timestamp","length":8,"nullable":true,"value":
["2010-01-01 12:30:00",null]},"bearing":{"type":"bool","length":1,"
nullable":true,"value":[null,true]}}}}'::trajectory a)
Select st_attrLength(a, 0) from traj;
 st_attrlength
---------------
             4
(1 row)

With traj AS (
  select '{"trajectory":{"version":1,"type":"STPOINT","leafcount":2,"
start_time":"2010-01-01 11:30:00","end_time":"2010-01-01 12:30:00","
spatial":"SRID=4326;LINESTRING(1 1,3 5)","timeline":["2010-01-01 11:30
:00","2010-01-01 12:30:00"],"attributes":{"leafcount":2,"velocity":{"
type":"integer","length":4,"nullable":true,"value":[1,null]},"speed":
{"type":"float","length":8,"nullable":true,"value":[null,1.0]},"angel
":{"type":"string","length":64,"nullable":true,"value":["test",null
]}, "tngel2":{"type":"timestamp","length":8,"nullable":true,"value":
["2010-01-01 12:30:00",null]},"bearing":{"type":"bool","length":1,"
nullable":true,"value":[null,true]}}}}'::trajectory a)
Select st_attrLength(a, 'velocity') from traj;
 st_attrlength
---------------
             4
```

```
(1 row)
```

6.4.6 ST_attrNullable

获得轨迹属性是否允许为空。

语法

```
bool ST_attrNullable(trajectory traj, integer index) ;
bool ST_attrNullable(trajectory traj, text name) ;
```

参数

参数名称	描述
traj	**轨迹对象。**
index	**索引轨迹项。**
name	**属性名称。**

示例

```
With traj AS (
  select '{"trajectory":{"version":1,"type":"STPOINT","leafcount":2,"
start_time":"2010-01-01 11:30:00","end_time":"2010-01-01 12:30:00","
spatial":"SRID=4326;LINESTRING(1 1,3 5)","timeline":["2010-01-01 11:30
:00","2010-01-01 12:30:00"],"attributes":{"leafcount":2,"velocity":{"
type":"integer","length":4,"nullable":true,"value":[1,null]},"speed":
{"type":"float","length":8,"nullable":true,"value":[null,1.0]},"angel
":{"type":"string","length":64,"nullable":true,"value":["test",null
]}, "tngel2":{"type":"timestamp","length":8,"nullable":true,"value":
["2010-01-01 12:30:00",null]},"bearing":{"type":"bool","length":1,"
nullable":true,"value":[null,true]}}}}'::trajectory a)
Select st_attrNullable(a, 'velocity') from traj;
 st_attrnullable
-----------------
 t
(1 row)

With traj AS (
  select '{"trajectory":{"version":1,"type":"STPOINT","leafcount":2,"
start_time":"2010-01-01 11:30:00","end_time":"2010-01-01 12:30:00","
spatial":"SRID=4326;LINESTRING(1 1,3 5)","timeline":["2010-01-01 11:30
:00","2010-01-01 12:30:00"],"attributes":{"leafcount":2,"velocity":{"
type":"integer","length":4,"nullable":true,"value":[1,null]},"speed":
{"type":"float","length":8,"nullable":true,"value":[null,1.0]},"angel
":{"type":"string","length":64,"nullable":true,"value":["test",null
]}, "tngel2":{"type":"timestamp","length":8,"nullable":true,"value":
["2010-01-01 12:30:00",null]},"bearing":{"type":"bool","length":1,"
nullable":true,"value":[null,true]}}}}'::trajectory a)
Select st_attrNullable(a, 1) from traj;
 st_attrnullable
-----------------
 t
```

```
(1 row)
```

6.5 事件函数

6.5.1 ST_addEvent

给轨迹增加一个事件。

语法

```
trajectory ST_addEvent(trajectory traj, integer event_type, timestamp
event_time) ;
```

参数

参数名称	描述
traj	**轨迹对象。**
event_type	**事件类型。**
event_time	**事件时间戳。**

描述

事件类型由用户预先定义好，如 ‘1000’ 表示开锁， ‘2000’表示关锁 等。

示例

```
With traj AS (
  select '{"trajectory":{"version":1,"type":"STPOINT","leafcount":2,"
start_time":"2010-01-01 11:30:00","end_time":"2010-01-01 12:30:00","
spatial":"SRID=4326;LINESTRING(1 1,3 5)","timeline":["2010-01-01 11:30
:00","2010-01-01 12:30:00"],"attributes":{"leafcount":2,"velocity":{"
type":"integer","length":4,"nullable":true,"value":[1,null]},"speed":
{"type":"float","length":8,"nullable":true,"value":[null,1.0]},"angel
":{"type":"string","length":64,"nullable":true,"value":["test",null
]}, "tngel2":{"type":"timestamp","length":8,"nullable":true,"value":
["2010-01-01 12:30:00",null]},"bearing":{"type":"bool","length":1,"
nullable":true,"value":[null,true]}}}}'::trajectory a)
Select st_addevent(a, 1, '2010-01-01 11:30:00' ) from traj;


                                                        st_addevent


-------------------------------------------------------------------------------
 {"trajectory":{"version":1,"type":"STPOINT","leafcount":2,"start_time
":"2010-01-01 11:30:00","end_time":"2010-01-01 12:30:00","spatial":"
SRID=4326;LINESTRING(1 1,3 5)","timeline":["2010-01-01 11:30:00","
```

```
2010-01-01 12:30:00"],"attributes":{"leafcount":2,"velocity":{"type
":"integer","length":4,"nullable":true,"value":[1,null]},"speed":{"
type":"float","length":8,"nullable":true,"value":[null,1.0]},"angel":
{"type":"string","length":64,"nullable":true,"value":["test",null]},"
tngel2":{"type":"timestamp","length":8,"nullable":true,"value":["2010-
01-01 12:30:00",null]},"bearing":{"type":"bool","length":1,"nullable":
true,"value":[null,true]}},"events":[{"1":"2010-01-01 11:30:00"}]}}
(1 row)
```

6.5.2 ST_eventTimes

获得轨迹的所有事件时间。

语法

```
timestamp[] ST_eventTimes(trajectory traj) ;
```

参数

参数名称	描述
traj	**轨迹对象。**

示例

```
With traj AS (
  select '{"trajectory":{"version":1,"type":"STPOINT","leafcount":2,"
start_time":"2010-01-01 11:30:00","end_time":"2010-01-01 12:30:00","
spatial":"SRID=4326;LINESTRING(1 1,3 5)","timeline":["2010-01-01 11:30
:00","2010-01-01 12:30:00"],"attributes":{"leafcount":2,"velocity":{"
type":"integer","length":4,"nullable":true,"value":[1,null]},"speed":
{"type":"float","length":8,"nullable":true,"value":[null,1.0]},"angel
":{"type":"string","length":64,"nullable":true,"value":["test",null
]}, "tngel2":{"type":"timestamp","length":8,"nullable":true,"value":
["2010-01-01 12:30:00",null]},"bearing":{"type":"bool","length":1,"
nullable":true,"value":[null,true]}},"events":[{"1":"Fri Jan 01 14:30:
00 2010"},{"2":"Fri Jan 01 14:30:00 2010"}]}}'::trajectory a)
Select st_eventTimes(a ) from traj;
                st_eventtimes
---------------------------------------------
 {"2010-01-01 14:30:00","2010-01-01 14:30:00"}
(1 row)
```

6.5.3 ST_eventTime

获得轨迹指定索引事件时间。

语法

```
timestamp ST_eventTime(trajectory traj, integer index) ;
```

参数

参数名称	描述
traj	**轨迹对象。**

参数名称	描述
index	事件索引序号。

示例

```
With traj AS (
  select '{"trajectory":{"version":1,"type":"STPOINT","leafcount":2,"
start_time":"2010-01-01 11:30:00","end_time":"2010-01-01 12:30:00","
spatial":"SRID=4326;LINESTRING(1 1,3 5)","timeline":["2010-01-01 11:30
:00","2010-01-01 12:30:00"],"attributes":{"leafcount":2,"velocity":{"
type":"integer","length":4,"nullable":true,"value":[1,null]},"speed":
{"type":"float","length":8,"nullable":true,"value":[null,1.0]},"angel
":{"type":"string","length":64,"nullable":true,"value":["test",null
]}, "tngel2":{"type":"timestamp","length":8,"nullable":true,"value":
["2010-01-01 12:30:00",null]},"bearing":{"type":"bool","length":1,"
nullable":true,"value":[null,true]}},"events":[{"1":"Fri Jan 01 14:30:
00 2010"}]}}'::trajectory a)
Select st_eventTime(a, 0 ) from traj;
    st_eventtime
---------------------
 2010-01-01 14:30:00
(1 row)
```

6.5.4 ST_eventTypes

获得轨迹的所有事件类型。

语法

```
integer[]  ST_eventTypes( trajectory traj) ;
```

参数

参数名称	描述
traj	轨迹对象。

示例

```
With traj AS (
  select '{"trajectory":{"version":1,"type":"STPOINT","leafcount":2,"
start_time":"2010-01-01 11:30:00","end_time":"2010-01-01 12:30:00","
spatial":"SRID=4326;LINESTRING(1 1,3 5)","timeline":["2010-01-01 11:30
:00","2010-01-01 12:30:00"],"attributes":{"leafcount":2,"velocity":{"
type":"integer","length":4,"nullable":true,"value":[1,null]},"speed":
{"type":"float","length":8,"nullable":true,"value":[null,1.0]},"angel
":{"type":"string","length":64,"nullable":true,"value":["test",null
]}, "tngel2":{"type":"timestamp","length":8,"nullable":true,"value":
["2010-01-01 12:30:00",null]},"bearing":{"type":"bool","length":1,"
nullable":true,"value":[null,true]}},"events":[{"1":"Fri Jan 01 14:30:
00 2010"},{"2":"Fri Jan 01 14:30:00 2010"}]}}'::trajectory a)
Select st_eventTypes(a ) from traj;
 st_eventtypes
---------------
```

```
 {1,2}
```

6.5.5 ST_eventType

获得轨迹指定索引事件的类型。

语法

```
integer ST_eventType(trajectory traj, integer index) ;
```

参数

参数名称	描述
traj	**轨迹对象。**
index	**事件序号。**

示例

```
With traj AS (
  select '{"trajectory":{"version":1,"type":"STPOINT","leafcount":2,"
start_time":"2010-01-01 11:30:00","end_time":"2010-01-01 12:30:00","
spatial":"SRID=4326;LINESTRING(1 1,3 5)","timeline":["2010-01-01 11:30
:00","2010-01-01 12:30:00"],"attributes":{"leafcount":2,"velocity":{"
type":"integer","length":4,"nullable":true,"value":[1,null]},"speed":
{"type":"float","length":8,"nullable":true,"value":[null,1.0]},"angel
":{"type":"string","length":64,"nullable":true,"value":["test",null
]}, "tngel2":{"type":"timestamp","length":8,"nullable":true,"value":
["2010-01-01 12:30:00",null]},"bearing":{"type":"bool","length":1,"
nullable":true,"value":[null,true]}},"events":[{"1":"Fri Jan 01 14:30:
00 2010"},{"2":"Fri Jan 01 14:30:00 2010"}]}}'::trajectory a)
Select st_eventType(a, 0 ) from traj;
 st_eventtype
--------------
            1
```

6.6 属性函数

6.6.1 ST_startTime

获得轨迹的起始时间。

语法

```
timestamp ST_startTime(trajectory traj) ;
```

参数

参数名称	描述
traj	**需要获得开始时间的轨迹。**

示例

```
Select ST_startTime(traj) From traj_table;
```

6.6.2 ST_endTime

获得轨迹的结束时间。

语法

```
timestamp ST_endTime(trajectory traj) ;
```

参数

参数名称	描述
traj	需要获得结束时间的轨迹。

示例

```
Select ST_endTime(traj) From traj_table;
```

6.6.3 ST_trajectorySpatial

获得轨迹的几何对象。

语法

```
geometry ST_trajectorySpatial(trajectory traj);
geometry ST_trajSpatial(trajectory traj);
```

参数

参数名称	描述
traj	需要获得几何对象的轨迹。

示例

```
Select AsText(ST_trajectorySpatial(traj)) FROM traj_table;
```

6.6.4 ST_trajectoryTemporal

获得轨迹的时间线。

语法

```
text ST_trajectoryTemporal(trajectory traj);
```

```
text ST_trajTemporal(trajectory traj);
```

参数

参数名称	描述
traj	需要获得时间线的轨迹。

示例

```
select ST_trajectoryTemporal(ST_MakeTrajectory('STPOINT'::leaftype,
st_geomfromtext('LINESTRING (114 35, 115 36, 116 37)', 4326), '[2010-
01-01 14:30, 2010-01-01 15:30)'::tsrange, null));
                               st_trajectorytemporal

--------------------------------------------------------------------------------
 {"timeline":["2010-01-01 14:30:00","2010-01-01 15:00:00","2010-01-01
15:30:00"]}
(1 row)
```

6.6.5 ST_trajAttrs

获得轨迹的属性信息。

语法

```
text  ST_trajAttrs(trajectory traj);
```

参数

参数名称	描述
traj	需要获得属性信息的轨迹。

示例

```
Select ST_trajAttrs(traj) From traj_table;
```

6.6.6 ST_attrIntMax

获得轨迹属性字段类型为integer的属性最大值。

语法

```
int8    ST_attrIntMax(trajectory traj, cstring attr_field_name);
```

参数

参数名称	描述
traj	需要获得属性信息的轨迹。
attr_field_name	指定的属性字段名称。

示例

```
select st_attrIntMax(ST_makeTrajectory('STPOINT'::leaftype, '
LINESTRING(-179.48077 51.72814,-179.46731 51.74634,-179.46502 51.74934
,-179.46183 51.75378,-179.45943 51.75736,-179.45560 51.76273,-179.
44845 51.77186,-179.43419 51.78977,-179.41259 51.81643,-179.41001 51.
81941,-179.40751 51.82223,-179.40497 51.82505,-179.40242 51.82796,-179
.39981 51.83095,-179.39734 51.83398,-179.39499 51.83709)'::geometry,
ARRAY['2017-01-15 09:06:39'::timestamp,'2017-01-15 09:14:48','2017-01
-15 09:13:39','2017-01-15 09:16:28','2017-01-15 09:19:48','2017-01-15
 09:17:48','2017-01-15 09:23:19','2017-01-15 09:34:40','2017-01-15 09
:30:28','2017-01-15 09:36:59','2017-01-15 09:38:09','2017-01-15 09:39
:18','2017-01-15 09:40:40','2017-01-15 09:47:38','2017-01-15 21:18:30
','2017-01-15 09:48:49'], '{"leafcount": 16, "attributes" : {"heading
" : {"type": "integer", "length": 4, "nullable" : false,"value":[0,1,2
,3,4,5,6,7,8,9,10,11,12,13,14,15]}}}'), 'heading');
 st_attrintmax
---------------
            15
```

6.6.7 ST_attrIntMin

获得轨迹属性字段类型为integer的属性最小值。

语法

```
int8    ST_attrIntMin(trajectory traj, cstring attr_field_name);
```

参数

参数名称	描述
traj	**需要获得属性信息的轨迹。**
attr_field_name	**指定的属性字段名称。**

示例

```
select st_attrIntMin(ST_makeTrajectory('STPOINT'::leaftype, '
LINESTRING(-179.48077 51.72814,-179.46731 51.74634,-179.46502 51.74934
,-179.46183 51.75378,-179.45943 51.75736,-179.45560 51.76273,-179.
44845 51.77186,-179.43419 51.78977,-179.41259 51.81643,-179.41001 51.
81941,-179.40751 51.82223,-179.40497 51.82505,-179.40242 51.82796,-179
.39981 51.83095,-179.39734 51.83398,-179.39499 51.83709)'::geometry,
ARRAY['2017-01-15 09:06:39'::timestamp,'2017-01-15 09:14:48','2017-01
-15 09:13:39','2017-01-15 09:16:28','2017-01-15 09:19:48','2017-01-15
 09:17:48','2017-01-15 09:23:19','2017-01-15 09:34:40','2017-01-15 09
:30:28','2017-01-15 09:36:59','2017-01-15 09:38:09','2017-01-15 09:39
:18','2017-01-15 09:40:40','2017-01-15 09:47:38','2017-01-15 21:18:30
','2017-01-15 09:48:49'], '{"leafcount": 16, "attributes" : {"heading
" : {"type": "integer", "length": 4, "nullable" : false,"value":[0,1,2
,3,4,5,6,7,8,9,10,11,12,13,14,15]}}}'), 'heading');
 st_attrintmin
---------------
```

```
              0
```

6.6.8 ST_attrIntAverage

获得轨迹属性字段类型为integer的属性平均值。

语法

```
int8    ST_attrIntAverage(trajectory traj, cstring attr_field_name);
```

参数

参数名称	描述
traj	**需要获得属性信息的轨迹。**
attr_field_name	**指定的属性字段名称。**

示例

```
select st_attrIntAverage(ST_makeTrajectory('STPOINT'::leaftype, '
LINESTRING(-179.48077 51.72814,-179.46731 51.74634,-179.46502 51.74934
,-179.46183 51.75378,-179.45943 51.75736,-179.45560 51.76273,-179.
44845 51.77186,-179.43419 51.78977,-179.41259 51.81643,-179.41001 51.
81941,-179.40751 51.82223,-179.40497 51.82505,-179.40242 51.82796,-179
.39981 51.83095,-179.39734 51.83398,-179.39499 51.83709)'::geometry,
ARRAY['2017-01-15 09:06:39'::timestamp,'2017-01-15 09:14:48','2017-01
-15 09:13:39','2017-01-15 09:16:28','2017-01-15 09:19:48','2017-01-15
 09:17:48','2017-01-15 09:23:19','2017-01-15 09:34:40','2017-01-15 09
:30:28','2017-01-15 09:36:59','2017-01-15 09:38:09','2017-01-15 09:39
:18','2017-01-15 09:40:40','2017-01-15 09:47:38','2017-01-15 21:18:30
','2017-01-15 09:48:49'], '{"leafcount": 16, "attributes" : {"heading
" : {"type": "integer", "length": 4, "nullable" : false,"value":[0,1,2
,3,4,5,6,7,8,9,10,11,12,13,14,15]}}}'), 'heading');
 st_attrintaverage
-------------------
                 7
```

6.6.9 ST_attrFloatMax

获得轨迹属性字段类型为float的属性最大值。

语法

```
float8    ST_attrFloatMax(trajectory traj, cstring attr_field_name);
```

参数

参数名称	描述
traj	**需要获得属性信息的轨迹。**
attr_field_name	**指定的属性字段名称。**

示例

```
select st_attrFloatMax(ST_makeTrajectory('STPOINT'::leaftype, '
LINESTRING(-179.48077 51.72814,-179.46731 51.74634,-179.46502 51.74934
,-179.46183 51.75378,-179.45943 51.75736,-179.45560 51.76273,-179.
44845 51.77186,-179.43419 51.78977,-179.41259 51.81643,-179.41001 51.
81941,-179.40751 51.82223,-179.40497 51.82505,-179.40242 51.82796,-179
.39981 51.83095,-179.39734 51.83398,-179.39499 51.83709)'::geometry,
ARRAY['2017-01-15 09:06:39'::timestamp,'2017-01-15 09:13:39','2017-01
-15 09:14:48','2017-01-15 09:16:28','2017-01-15 09:17:48','2017-01-15
 09:19:48','2017-01-15 09:23:19','2017-01-15 09:30:28','2017-01-15 09
:34:40','2017-01-15 09:36:59','2017-01-15 09:38:09','2017-01-15 09:39
:18','2017-01-15 09:40:40','2017-01-15 09:47:38','2017-01-15 09:48:49
','2017-01-15 21:18:30'], '{"leafcount": 16, "attributes" : {"heading
" : {"type": "float", "length": 4, "nullable" : false,"value": [23.0,
23.0,23.0,23.0,21.0,21.0,72.0,72.0,72.0,72.0,73.0,74.0,73.0,73.0,73.0,
73.0]}}}' ),'heading');
 st_attrfloatmax
-----------------
              74
```

6.6.10 ST_attrFloatMin

获得轨迹属性字段类型为float的属性最小值。

语法

```
float8    ST_attrFloatMax(trajectory traj, cstring attr_field_name);
```

参数

参数名称	描述
traj	需要获得属性信息的轨迹。
attr_field_name	指定的属性字段名称。

示例

```
select st_attrFloatMin(ST_makeTrajectory('STPOINT'::leaftype, '
LINESTRING(-179.48077 51.72814,-179.46731 51.74634,-179.46502 51.74934
,-179.46183 51.75378,-179.45943 51.75736,-179.45560 51.76273,-179.
44845 51.77186,-179.43419 51.78977,-179.41259 51.81643,-179.41001 51.
81941,-179.40751 51.82223,-179.40497 51.82505,-179.40242 51.82796,-179
.39981 51.83095,-179.39734 51.83398,-179.39499 51.83709)'::geometry,
ARRAY['2017-01-15 09:06:39'::timestamp,'2017-01-15 09:13:39','2017-01
-15 09:14:48','2017-01-15 09:16:28','2017-01-15 09:17:48','2017-01-15
 09:19:48','2017-01-15 09:23:19','2017-01-15 09:30:28','2017-01-15 09
:34:40','2017-01-15 09:36:59','2017-01-15 09:38:09','2017-01-15 09:39
:18','2017-01-15 09:40:40','2017-01-15 09:47:38','2017-01-15 09:48:49
','2017-01-15 21:18:30'], '{"leafcount": 16, "attributes" : {"heading
" : {"type": "float", "length": 4, "nullable" : false,"value": [23.0,
23.0,23.0,23.0,21.0,21.0,72.0,72.0,72.0,72.0,73.0,74.0,73.0,73.0,73.0,
73.0]}}}' ),'heading');
 st_attrfloatmin
-----------------
```

```
                21
```

6.6.11 ST_attrFloatAverage

获得轨迹属性字段类型为float的属性平均值。

语法

```
float8    ST_attrFloatAverage(trajectory traj, cstring attr_field_name
);
```

参数

参数名称	描述
traj	**需要获得属性信息的轨迹。**
attr_field_name	**指定的属性字段名称。**

示例

```
select st_attrFloatAverage(ST_makeTrajectory('STPOINT'::leaftype, '
LINESTRING(-179.48077 51.72814,-179.46731 51.74634,-179.46502 51.74934
,-179.46183 51.75378,-179.45943 51.75736,-179.45560 51.76273,-179.
44845 51.77186,-179.43419 51.78977,-179.41259 51.81643,-179.41001 51.
81941,-179.40751 51.82223,-179.40497 51.82505,-179.40242 51.82796,-179
.39981 51.83095,-179.39734 51.83398,-179.39499 51.83709)'::geometry,
ARRAY['2017-01-15 09:06:39'::timestamp,'2017-01-15 09:13:39','2017-01
-15 09:14:48','2017-01-15 09:16:28','2017-01-15 09:17:48','2017-01-15
 09:19:48','2017-01-15 09:23:19','2017-01-15 09:30:28','2017-01-15 09
:34:40','2017-01-15 09:36:59','2017-01-15 09:38:09','2017-01-15 09:39
:18','2017-01-15 09:40:40','2017-01-15 09:47:38','2017-01-15 09:48:49
','2017-01-15 21:18:30'], '{"leafcount": 16, "attributes" : {"heading
" : {"type": "float", "length": 4, "nullable" : false,"value": [23.0,
23.0,23.0,23.0,21.0,21.0,72.0,72.0,72.0,72.0,73.0,74.0,73.0,73.0,73.0,
73.0]}}}' ),'heading');
 st_attrfloataverage
---------------------
             53.8125
(1 row)
```

6.6.12 ST_leafType

获得轨迹的叶面类型。

语法

```
text ST_leafType(trajectory traj);
```

参数

参数名称	描述
traj	**需要获得类型的轨迹。**

目前只支持ST_POINT类型。

示例

```
select st_leafType(traj) from traj where id = 3;
 st_leaftype
 -------------
STPOINT
(1 row)
```

6.6.13 ST_leafCount

获得轨迹的叶子数量（轨迹点个数）。

语法

```
integer ST_leafCount(trajectory traj);
```

参数

参数名称	描述
traj	需要获得类型的轨迹。

示例

```
select st_leafCount(traj) from traj where id = 2;
 st_leafcount
--------------
16
```

6.6.14 ST_duration

获得该轨迹的持续时间。

语法

```
interval ST_uration(trajectory traj);
```

参数

参数名称	描述
traj	轨迹数据。

示例

```
select st_duration(traj) from traj where id = 2;
 st_duration
-------------
 00:42:10
```

```
(1 row)
```

6.6.15 ST_Distance（下线）

获取点类型轨迹的轨迹总长度。

语法

```
float8 ST_Distance(trajectory traj);
```

参数

参数名称	描述
traj	**轨迹数据。**

示例

```
Select ST_Distance(traj) from traj_table;
```

6.6.16 ST_timeAtPoint

获取移动轨迹通过某位置的时间点集合。

语法

```
timestamp[] ST_timeAtPoint(trajectory traj, geomery g);
```

参数

参数名称	描述
traj	**轨迹对象。**
g	**点位置。**

6.6.17 ST_pointAtTime

获取指定时间位置的几何对象。

语法

```
geometry ST_pointAtTime(trajectory traj, timestamp t);
```

参数

参数名称	描述
traj	**轨迹对象。**
t	**指定的时间点。**

返回点对象。

示例

```
Select ST_pointAtTime(traj, '2010-1-11 23:40:00') from traj_table;
```

6.6.18 ST_velocityAtTime

获取指定时间位置的速度属性。

语法

```
float8 ST_ VelocityAtTime (trajectory traj, timestamp t);
```

参数

参数名称	描述
traj	**轨迹对象。**
t	**指定的时间点。**

示例

```
SeSelect ST_velocityAtTime(traj) From traj_table;
```

6.6.19 ST_accelerationAtTime

获取指定时间位置的加速度属性。

语法

```
float8 ST_accelerationAtTime (trajectory traj, timestamp t);
```

参数

参数名称	描述
traj	**轨迹对象。**
t	**指定的时间点。**

示例

```
Select ST_accelerationAtTime(traj) From traj_table;
```

6.6.20 ST_bearingAtTime（下线）

获取指定时间位置的方位角属性。

语法

```
float8 ST_bearingAtTime (trajectory traj, timestamp t);
```

参数

参数名称	描述
traj	**轨迹对象。**
t	**指定的时间点。**

示例

```
Select ST_bearingAtTime(traj) From traj_table;
```

6.6.21 ST_accuraryAtTime（下线）

获取指定时间位置的精度属性。

语法

```
float8 ST_accuraryAtTime (trajectory traj, timestamp t);
```

参数

参数名称	描述
traj	**轨迹对象。**
t	**指定的时间点。**

示例

```
Select ST_accuraryAtTime(traj) From traj_table;
```

6.6.22 ST_timeToDistance

输出时间到欧氏距离的函数，以折线输出，横坐标为时间，纵坐标为距离。

语法

```
geometry ST_timeToDistance(trajectory traj);
```

参数

参数名称	描述
traj	**轨迹对象。**

示例

```
Select ST_timeToDistance(traj) from traj_table;
```

6.6.23 ST_timeAtDistance

从起始点移动指定距离后所在的时间点。

语法

```
timestamp[] ST_timeAtDistance(trajectory traj, float8 d);
```

参数

参数名称	描述
traj	**轨迹对象。**
d	**距离。**

返回的是 timestamp的数组，有可能存在多个点到起始点的距离一致。

示例

```
Select ST_timeAtDistance(traj, 100) from traj_table;
```

6.6.24 ST_cumulativeDistanceAtTime

从起点出发到指定时间点累计的位移长度。

语法

```
float8 ST_cumulativeDistanceAtTime(trajectory traj, timestamp t)
```

```
;
```

参数

参数名称	描述
traj	轨迹对象。
t	时间点。

示例

```
Select ST_cumulativeDistanceAtTime(traj, '2011-11-1 04:30:00') from
traj_table;
```

6.6.25 ST_timeAtCumulativeDistance

从起点出发位移到指定长度时所处的时间点。

语法

```
timestamp ST_timeAtCumulativeDistance(trajectory traj, float d)
;
```

参数

参数名称	描述
traj	轨迹对象。
d	累计长度。

示例

```
Select ST_timeAtCumulativeDistance(traj, 100.0) from traj_table;
```

6.6.26 ST_subTrajectory

根据时间段截取子段。

语法

```
trajectory ST_subTrajectory(trajectory traj, timestamp starttime,
timestamp endtime) ;

trajectory ST_subTrajectory(trajectory traj, tsrange range);
```

参数

参数名称	描述
traj	轨迹对象。

参数名称	描述
starttime	开始时间。
endtime	结束时间。
range	时间段。

示例

```
Select ST_subTrajectory(traj, '2010-1-11 02:45:30', '2010-1-11 03:00:
00') FROM traj_table;
```

6.6.27 ST_subTrajectorySpatial

指定时间段的轨迹几何。

语法

```
geometry ST_subTrajectorySpatial(trajectory traj, timestamp starttime
, timestamp endtime) ;
geometry ST_subTrajectorySpatial(trajectory traj, tsrange range) ;
```

参数

参数名称	描述
traj	轨迹对象。
starttime	开始时间。
endtime	结束时间。
range	时间段。

示例

```
Select ST_subTrajectorySpatial(traj, '2010-1-11 02:45:30', '2010-1-11
03:00:00') FROM traj_table;
```

6.6.28 ST_samplingInterval

获取采样间隔。

语法

```
iinterval ST_samplingInterval(trajectory traj);
```

参数

参数名称	描述
traj	**轨迹对象。**

示例

```
select ST_samplingInterval(ST_MakeTrajectory('STPOINT'::leaftype,
st_geomfromtext('LINESTRING (114 35, 115 36, 116 37)', 4326), '[2010-
01-01 14:30, 2010-01-01 15:30)'::tsrange, '{"leafcount":3,"attributes
":{"velocity": {"type": "integer", "length": 2,"nullable" : true,"
value": [120, 130, 140]}, "accuracy": {"type": "float", "length": 4
, "nullable" : false,"value": [120, 130, 140]}, "bearing": {"type":
 "float", "length": 8, "nullable" : false,"value": [120, 130, 140]},
 "acceleration": {"type": "string", "length": 20, "nullable" : true
,"value": ["120", "130", "140"]}, "active": {"type": "timestamp", "
nullable" : false,"value": ["Fri Jan 01 14:30:00 2010", "Fri Jan 01
15:00:00 2010", "Fri Jan 01 15:30:00 2010"]}}, "events": [{"1" : "Fri
 Jan 01 14:30:00 2010"}, {"2" : "Fri Jan 01 15:00:00 2010"}, {"3" : "
Fri Jan 01 15:30:00 2010"}]}'));
 st_samplinginterval
---------------------
 @ 20 mins
(1 row)
```

6.6.29 ST_trajAttrsAsText

获得作为文本类型的轨迹属性数组。

语法

```
text[]  st_trajAttrsAsText(trajectory traj, text attr_name);
```

参数

参数名称	描述
traj	**轨迹对象。**
attr_name	**属性名称。**

示例

```
With traj AS (
  select '{"trajectory":{"version":1,"type":"STPOINT","leafcount":2,"
start_time":"2010-01-01 11:30:00","end_time":"2010-01-01 12:30:00","
spatial":"SRID=4326;LINESTRING(1 1,3 5)","timeline":["2010-01-01 11:30
:00","2010-01-01 12:30:00"],"attributes":{"leafcount":2,"velocity":{"
type":"integer","length":4,"nullable":true,"value":[1,null]},"speed":
{"type":"float","length":8,"nullable":true,"value":[null,1.0]},"angel
":{"type":"string","length":64,"nullable":true,"value":["test",null
]}, "tngel2":{"type":"timestamp","length":8,"nullable":true,"value":
["2010-01-01 12:30:00",null]},"bearing":{"type":"bool","length":1,"
nullable":true,"value":[null,true]}}}}'::trajectory a)
Select st_trajAttrsAsText(a, 'angel') from traj;
 st_trajattrsastext
--------------------
 {test,NULL}
(1 row)

With traj AS (
  select '{"trajectory":{"version":1,"type":"STPOINT","leafcount":2,"
start_time":"2010-01-01 11:30:00","end_time":"2010-01-01 12:30:00","
spatial":"SRID=4326;LINESTRING(1 1,3 5)","timeline":["2010-01-01 11:30
:00","2010-01-01 12:30:00"],"attributes":{"leafcount":2,"velocity":{"
type":"integer","length":4,"nullable":true,"value":[1,null]},"speed":
{"type":"float","length":8,"nullable":true,"value":[null,1.0]},"angel
":{"type":"string","length":64,"nullable":true,"value":["test",null
]}, "tngel2":{"type":"timestamp","length":8,"nullable":true,"value":
["2010-01-01 12:30:00",null]},"bearing":{"type":"bool","length":1,"
nullable":true,"value":[null,true]}}}}'::trajectory a)
Select st_trajAttrsAsText(a, 'tngel2') from traj;
      st_trajattrsastext
-------------------------------
 {"2010-01-01 12:30:00",NULL}
(1 row)

With traj AS (
  select '{"trajectory":{"version":1,"type":"STPOINT","leafcount":2,"
start_time":"2010-01-01 11:30:00","end_time":"2010-01-01 12:30:00","
spatial":"SRID=4326;LINESTRING(1 1,3 5)","timeline":["2010-01-01 11:30
:00","2010-01-01 12:30:00"],"attributes":{"leafcount":2,"velocity":{"
type":"integer","length":4,"nullable":true,"value":[1,null]},"speed":
{"type":"float","length":8,"nullable":true,"value":[null,1.0]},"angel
":{"type":"string","length":64,"nullable":true,"value":["test",null
]}, "tngel2":{"type":"timestamp","length":8,"nullable":true,"value":
["2010-01-01 12:30:00",null]},"bearing":{"type":"bool","length":1,"
nullable":true,"value":[null,true]}}}}'::trajectory a)
Select st_trajAttrsAsText(a, 'bearing') from traj;
 st_trajattrsastext
--------------------
 {NULL,t}
```

```
(1 row)
```

6.6.30 ST_trajAttrsAsInteger

获得作为文本类型的轨迹属性数组。

语法

```
integer[]  st_trajAttrsAsInteger(trajectory traj, text attr_name);
```

参数

参数名称	描述
traj	**轨迹对象。**
attr_name	**属性名称。**

示例

```
With traj AS (
  select '{"trajectory":{"version":1,"type":"STPOINT","leafcount":2,"
start_time":"2010-01-01 11:30:00","end_time":"2010-01-01 12:30:00","
spatial":"SRID=4326;LINESTRING(1 1,3 5)","timeline":["2010-01-01 11:30
:00","2010-01-01 12:30:00"],"attributes":{"leafcount":2,"velocity":{"
type":"integer","length":4,"nullable":true,"value":[1,null]},"speed":
{"type":"float","length":8,"nullable":true,"value":[null,1.0]},"angel
":{"type":"string","length":64,"nullable":true,"value":["test",null
]}, "tngel2":{"type":"timestamp","length":8,"nullable":true,"value":
["2010-01-01 12:30:00",null]},"bearing":{"type":"bool","length":1,"
nullable":true,"value":[null,true]}}}}'::trajectory a)
Select st_trajAttrsAsInteger(a, 'speed') from traj;
 st_trajattrsasinteger
-----------------------
 {NULL,1}
(1 row)
```

6.6.31 ST_trajAttrsAsDouble

获得作为文本类型的轨迹属性数组。

语法

```
float8[]  st_trajAttrsAsDouble(trajectory traj, text attr_name);
```

参数

参数名称	描述
traj	**轨迹对象。**
attr_name	**属性名称。**

示例

```
With traj AS (
  select '{"trajectory":{"version":1,"type":"STPOINT","leafcount":2,"
start_time":"2010-01-01 11:30:00","end_time":"2010-01-01 12:30:00","
spatial":"SRID=4326;LINESTRING(1 1,3 5)","timeline":["2010-01-01 11:30
:00","2010-01-01 12:30:00"],"attributes":{"leafcount":2,"velocity":{"
type":"integer","length":4,"nullable":true,"value":[1,null]},"speed":
{"type":"float","length":8,"nullable":true,"value":[null,1.0]},"angel
":{"type":"string","length":64,"nullable":true,"value":["test",null
]}, "tngel2":{"type":"timestamp","length":8,"nullable":true,"value":
["2010-01-01 12:30:00",null]},"bearing":{"type":"bool","length":1,"
nullable":true,"value":[null,true]}}}}'::trajectory a)
Select st_trajAttrsAsDouble(a, 'velocity') from traj;
 st_trajattrsasdouble
----------------------
 {1,NULL}
(1 row)
```

6.6.32 ST_trajAttrsAsBool

获得作为文本类型的轨迹属性数组。

语法

```
bool[]  st_trajAttrsAsBool(trajectory traj, text attr_name);
```

参数

参数名称	描述
traj	**轨迹对象。**
attr_name	**属性名称。**

示例

```
With traj AS (
  select '{"trajectory":{"version":1,"type":"STPOINT","leafcount":2,"
start_time":"2010-01-01 11:30:00","end_time":"2010-01-01 12:30:00","
spatial":"SRID=4326;LINESTRING(1 1,3 5)","timeline":["2010-01-01 11:30
:00","2010-01-01 12:30:00"],"attributes":{"leafcount":2,"velocity":{"
type":"integer","length":4,"nullable":true,"value":[1,null]},"speed":
{"type":"float","length":8,"nullable":true,"value":[null,1.0]},"angel
":{"type":"string","length":64,"nullable":true,"value":["test",null
]}, "tngel2":{"type":"timestamp","length":8,"nullable":true,"value":
["2010-01-01 12:30:00",null]},"bearing":{"type":"bool","length":1,"
nullable":true,"value":[null,true]}}}}'::trajectory a)
Select st_trajAttrsAsBool(a, 'velocity') from traj;
 st_trajattrsasbool
--------------------
 {t,NULL}
```

```
(1 row)
```

6.6.33 ST_trajAttrsAsTimestamp

获得作为时间戳类型的轨迹属性数组。

语法

```
timestamp[]  st_trajAttrsAsTimestamp(trajectory traj, text attr_name);
```

参数

参数名称	描述
traj	**轨迹对象。**
attr_name	**属性名称。**

示例

```
With traj AS (
  select '{"trajectory":{"version":1,"type":"STPOINT","leafcount":2,"
start_time":"2010-01-01 11:30:00","end_time":"2010-01-01 12:30:00","
spatial":"SRID=4326;LINESTRING(1 1,3 5)","timeline":["2010-01-01 11:30
:00","2010-01-01 12:30:00"],"attributes":{"leafcount":2,"velocity":{"
type":"integer","length":4,"nullable":true,"value":[1,null]},"speed":
{"type":"float","length":8,"nullable":true,"value":[null,1.0]},"angel
":{"type":"string","length":64,"nullable":true,"value":["test",null
]}, "tngel2":{"type":"timestamp","length":8,"nullable":true,"value":
["2010-01-01 12:30:00",null]},"bearing":{"type":"bool","length":1,"
nullable":true,"value":[null,true]}}}}'::trajectory a)
Select st_trajAttrsAsTimestamp(a, 'tngel2') from traj;
   st_trajattrsastimestamp
-----------------------------
 {"2010-01-01 12:30:00",NULL}
```

6.6.34 ST_attrIntFilter

指定轨迹属性字段，根据固定值，过滤出符合条件的轨迹点，返回过滤后的属性字段值（数组）。

语法

```
int8[] ST_attrIntFilter(trajectory traj, cstring attr_field_name,
cstring operator, int8 value);
int8[] ST_attrIntFilter(trajectory traj, cstring attr_field_name,
cstring operator, int8 value1, int8 value2);
```

参数

参数名称	描述
traj	**轨迹对象attr。**
attr_field_name	**指定的属性名称。**

参数名称	描述
operator	过滤符'=','!=','>', '<', '>=', '<=', '[]', '(]', '[)', '()'.
value、value1	属性固定值、下限。
value2	属性固定值-上限

描述

只支持类型为integer的属性字段。

示例

```
create table traj(id integer, traj trajectory);
insert into traj values(ST_makeTrajectory('STPOINT'::leaftype, '
LINESTRING(-179.48077 51.72814,-179.46731 51.74634,-179.46502 51.74934
,-179.46183 51.75378,-179.45943 51.75736,-179.45560 51.76273,-179.
44845 51.77186,-179.43419 51.78977,-179.41259 51.81643,-179.41001 51.
81941,-179.40751 51.82223,-179.40497 51.82505,-179.40242 51.82796,-179
.39981 51.83095,-179.39734 51.83398,-179.39499 51.83709)'::geometry,
ARRAY['2017-01-15 09:06:39'::timestamp,'2017-01-15 09:14:48','2017-01
-15 09:13:39','2017-01-15 09:16:28','2017-01-15 09:19:48','2017-01-15
 09:17:48','2017-01-15 09:23:19','2017-01-15 09:34:40','2017-01-15 09
:30:28','2017-01-15 09:36:59','2017-01-15 09:38:09','2017-01-15 09:39
:18','2017-01-15 09:40:40','2017-01-15 09:47:38','2017-01-15 21:18:30
','2017-01-15 09:48:49'], '{"leafcount": 16, "attributes" : {"heading
" : {"type": "integer", "length": 4, "nullable" : false,"value":[0,1,2
,3,4,5,6,7,8,9,10,11,12,13,14,15]}}}'));


select st_attrIntFilter(traj, 'heading', '>', 5) from traj where id =
1;
      st_attrintfilter
-----------------------------
 {6,7,8,9,10,11,12,13,14,15}
(1 row)

select st_attrIntFilter(traj, 'heading', '>=', 5) from traj where id
 = 1;
       st_attrintfilter
-------------------------------
 {5,6,7,8,9,10,11,12,13,14,15}
(1 row)

select st_attrIntFilter(traj, 'heading', '<', 5) from traj where id =
1;
 st_attrintfilter
------------------
 {0,1,2,3,4}
(1 row)

select st_attrIntFilter(traj, 'heading', '<=', 5) from traj where id
 = 1;
 st_attrintfilter
------------------
 {0,1,2,3,4,5}
(1 row)
```

```
select st_attrIntFilter(traj, 'heading', '=', 5) from traj where id =
1;
 st_attrintfilter
------------------
 {5}
(1 row)

select st_attrIntFilter(traj, 'heading', '!=', 5) from traj where id
 = 1;
           st_attrintfilter
---------------------------------------
 {0,1,2,3,4,6,7,8,9,10,11,12,13,14,15}
(1 row)

select st_attrIntFilter(traj, 'heading', '()', 5,8) from traj where id
 = 1;
 st_attrintfilter
------------------
 {6,7}
(1 row)

select st_attrIntFilter(traj, 'heading', '(]', 5,8) from traj where id
 = 1;
 st_attrintfilter
------------------
 {6,7,8}
(1 row)

select st_attrIntFilter(traj, 'heading', '[)', 5,8) from traj where id
 = 1;
 st_attrintfilter
------------------
 {5,6,7}
(1 row)

select st_attrIntFilter(traj, 'heading', '[]', 5,8) from traj where id
 = 1;
 st_attrintfilter
------------------
 {5,6,7,8}
(1 row)
```

6.6.35 ST_attrFloatFilter

指定轨迹属性字段，根据固定值，过滤出符合条件的轨迹点，返回过滤后的属性字段值（数组）。

语法

```
float8[] ST_attrFloatFilter(trajectory traj, cstring attr_field_name,
cstring operator, float8 value);
float8[] ST_attrFloatFilter(trajectory traj, cstring attr_field_name,
cstring operator, float8 value1, float8 value2);
```

参数

参数名称	描述
traj	**轨迹对象。**
attr_field_name	**指定的属性名称。**

参数名称	描述
operator	过滤符'=','!=','>', '<', '>=', '<=', '[]', '(]', '[)', '()'.
value、value1	属性固定值、下限。
value2	属性固定值-上限

描述

只支持类型为float的属性字段。

示例

```
create table traj(id integer, traj trajectory);
insert into traj values(2,ST_makeTrajectory('STPOINT'::leaftype, '
LINESTRING(-179.48077 51.72814,-179.46731 51.74634,-179.46502 51.74934
,-179.46183 51.75378,-179.45943 51.75736,-179.45560 51.76273,-179.
44845 51.77186,-179.43419 51.78977,-179.41259 51.81643,-179.41001 51.
81941,-179.40751 51.82223,-179.40497 51.82505,-179.40242 51.82796,-179
.39981 51.83095,-179.39734 51.83398,-179.39499 51.83709)'::geometry,
ARRAY['2017-01-15 09:06:39'::timestamp,'2017-01-15 09:14:48','2017-01
-15 09:13:39','2017-01-15 09:16:28','2017-01-15 09:19:48','2017-01-15
 09:17:48','2017-01-15 09:23:19','2017-01-15 09:34:40','2017-01-15 09
:30:28','2017-01-15 09:36:59','2017-01-15 09:38:09','2017-01-15 09:39
:18','2017-01-15 09:40:40','2017-01-15 09:47:38','2017-01-15 21:18:30
','2017-01-15 09:48:49'], '{"leafcount": 16, "attributes" : {"heading
" : {"type": "float", "length": 8, "nullable" : false,"value":[23.0,23
.0,23.0,23.0,21.0,21.0,72.0,72.0,72.0,72.0,73.0,74.0,73.0,73.0,73.0,73
.0]}}}'));


select st_attrFloatFilter(traj, 'heading', '>', 23.0) from traj where
id = 2;
       st_attrfloatfilter
---------------------------------
 {72,72,72,72,73,74,73,73,73,73}
(1 row)
```

6.6.36 ST_attrTimestampFilter

指定轨迹属性字段，根据固定值，过滤出符合条件的轨迹点，返回过滤后的属性字段值（数组）。

语法

```
timestamp[] ST_attrTimestampFilter(trajectory traj, cstring attr_field
_name,cstring operator, timestamp value);
timestamp[] ST_attrTimestampFilter(trajectory traj, cstring attr_field
_name,cstring operator, timestamp value1, timestamp value2);
```

参数

参数名称	描述
traj	轨迹对象。

参数名称	描述
attr_field_name	指定的属性名称。
operator	过滤符'=','!=','>', '<', '>=', '<=', '[]', '(]', '[)', '()'.
value、value1	属性固定值、下限。
value2	属性固定值-上限

示例

```
insert into traj values(3,ST_makeTrajectory('STPOINT'::leaftype,
st_geomfromtext('LINESTRING (114 35, 115 36, 116 37)', 4326), ARRAY['
2010-01-01 14:30'::timestamp, '2010-01-01 15:00', '2010-01-01 15:30'],
 '{"leafcount": 3, "attributes" : {"heading" : {"type": "timestamp", "
nullable" : false,"value":["Fri Jan 01 14:30:00 2010", "Fri Jan 01 15:
00:00 2010", "Fri Jan 01 15:30:00 2010"]}}}'));
select st_attrTimestampFilter(traj, 'heading', '>', 'Fri Jan 01 15:00:
00 2010'::timestamp) from traj where id = 3;
 st_attrtimestampfilter
------------------------
{"2010-01-01 15:30:00"}
(1 row)
```

6.6.37 ST_attrNullFilter

指定轨迹属性字段，过滤出值为空的轨迹点，返回由这些轨迹点构成的新轨迹。

语法

```
trajectory ST_attrNullFilter(trajectory traj, cstring attr_field_name
);
trajectory ST_attrNullFilter(trajectory traj, cstring attr_field_name
);
```

参数

参数名称	描述
traj	轨迹对象。
attr_field_name	指定的属性名称。

描述

支持所有类型的属性字段。

示例

```
select st_attrNullFilter(ST_makeTrajectory('STPOINT'::leaftype, '
LINESTRING(-179.48077 51.72814,-179.46731 51.74634,-179.46502 51.74934
,-179.46183 51.75378,-179.45943 51.75736,-179.45560 51.76273,-179.
44845 51.77186,-179.43419 51.78977,-179.41259 51.81643,-179.41001 51.
81941,-179.40751 51.82223,-179.40497 51.82505,-179.40242 51.82796,-179
```

```
.39981 51.83095,-179.39734 51.83398,-179.39499 51.83709)'::geometry,
ARRAY['2017-01-15 09:06:39'::timestamp,'2017-01-15 09:13:39','2017-01
-15 09:14:48','2017-01-15 09:16:28','2017-01-15 09:17:48','2017-01-15
 09:19:48','2017-01-15 09:23:19','2017-01-15 09:30:28','2017-01-15 09
:34:40','2017-01-15 09:36:59','2017-01-15 09:38:09','2017-01-15 09:39
:18','2017-01-15 09:40:40','2017-01-15 09:47:38','2017-01-15 09:48:49
','2017-01-15 21:18:30'], '{"leafcount": 16, "attributes" : {"heading
" : {"type": "float", "length": 4, "nullable" : true,"value": [23.0,23
.0,23.0,null,21.0,21.0,null,72.0,72.0,null,73.0,74.0,73.0,73.0,null,73
.0]}}}' ),'heading');


                                st_attrnullfilter


-------------------------------------------------------------------------------
 {"trajectory":{"version":1,"type":"STPOINT","leafcount":4,"start_time
":"2017-01-15 09:16:28","end_time":"2017-01-15 09:48:49","spatial":"
LINESTRING(-179.46183 51.75378,-179.44845 51.77
186,-179.41001 51.81941,-179.39734 51.83398)","timeline":["2017-01-15
 09:16:28","2017-01-15 09:23:19","2017-01-15 09:36:59","2017-01-15 09:
48:49"],"attributes":{"leafcount":4,"heading":
{"type":"float","length":4,"nullable":true,"value":[null,null,null,
null]}}}}
(1 row)
```

6.6.38 ST_attrNotNullFilter

指定轨迹属性字段，过滤出值不为空的轨迹点，返回由这些轨迹点构成的新轨迹。

语法

```
trajectory ST_attrNotNullFilter(trajectory traj, cstring attr_field
_name);
trajectory ST_attrNotNullFilter(trajectory traj, cstring attr_field
_name);
```

参数

参数名称	描述
traj	**轨迹对象。**
attr_field_name	**指定的属性名称。**

描述

支持所有类型的属性字段。

示例

```
select st_attrNotNullFilter(ST_makeTrajectory('STPOINT'::leaftype, '
LINESTRING(-179.48077 51.72814,-179.46731 51.74634,-179.46502 51.74934
,-179.46183 51.75378,-179.45943 51.75736,-179.45560 51.76273,-179.
44845 51.77186,-179.43419 51.78977,-179.41259 51.81643,-179.41001 51.
81941,-179.40751 51.82223,-179.40497 51.82505,-179.40242 51.82796,-179
```

```
.39981 51.83095,-179.39734 51.83398,-179.39499 51.83709)'::geometry,
ARRAY['2017-01-15 09:06:39'::timestamp,'2017-01-15 09:13:39','2017-01
-15 09:14:48','2017-01-15 09:16:28','2017-01-15 09:17:48','2017-01-15
 09:19:48','2017-01-15 09:23:19','2017-01-15 09:30:28','2017-01-15 09
:34:40','2017-01-15 09:36:59','2017-01-15 09:38:09','2017-01-15 09:39
:18','2017-01-15 09:40:40','2017-01-15 09:47:38','2017-01-15 09:48:49
','2017-01-15 21:18:30'], '{"leafcount": 16, "attributes" : {"heading
" : {"type": "float", "length": 4, "nullable" : true,"value": [23.0,23
.0,23.0,null,21.0,21.0,null,72.0,72.0,null,73.0,74.0,73.0,73.0,null,73
.0]}}}' ),'heading');


                                st_attrnotnullfilter


-------------------------------------------------------------------------------
 {"trajectory":{"version":1,"type":"STPOINT","leafcount":12,"
start_time":"2017-01-15 09:06:39","end_time":"2017-01-15 21:18:30","
spatial":"LINESTRING(-179.48077 51.72814,-179.46731 51.7
4634,-179.46502 51.74934,-179.45943 51.75736,-179.4556 51.76273,-179.
43419 51.78977,-179.41259 51.81643,-179.40751 51.82223,-179.40497 51.
82505,-179.40242 51.82796,-179.39981 51.83095,-
179.39499 51.83709)","timeline":["2017-01-15 09:06:39","2017-01-15 09
:13:39","2017-01-15 09:14:48","2017-01-15 09:17:48","2017-01-15 09:19:
48","2017-01-15 09:30:28","2017-01-15 09:34:40
","2017-01-15 09:38:09","2017-01-15 09:39:18","2017-01-15 09:40:40","
2017-01-15 09:47:38","2017-01-15 21:18:30"],"attributes":{"leafcount":
12,"heading":{"type":"float","length":4,"nulla
ble":true,"value":[23.0,23.0,23.0,21.0,21.0,72.0,72.0,73.0,74.0,73.0,
73.0,73.0]}}}}
(1 row)
```

6.6.39 ST_trajAttrsMeanMax

根据MEAN-MAX算法，计算出每个时间段内的均值的最大值。

语法

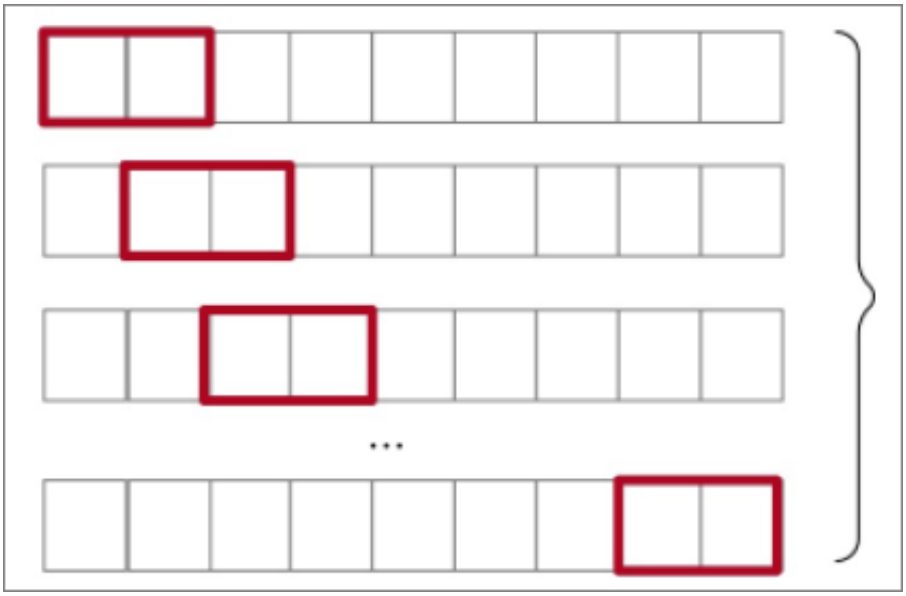
```
SETOF recrod ST_trajAttrsMeanMax(trajectory traj, cstring attr_field
_name, out interval duration, out float8 max);
```

参数

参数名称	描述
traj	**轨迹对象。**
attr_field_name	**指定的属性名称。**

描述

Mean-Max 算法通过一个滑动窗口，分别计算出落入该窗口的属性值的平均值，再求出所有均值的最大值。



该函数仅对integer和float类型的属性值有效。属性值不能为NULL。

示例

```
With traj AS (
    Select ST_makeTrajectory('STPOINT', 'LINESTRING(1 1, 6 6, 9 8, 10
12)'::geometry,
    ARRAY['2010-01-01 11:30'::timestamp, '2010-01-01 12:30', '2010-01-
01 13:30', '2010-01-01 14:30'],
    '{"leafcount":4, "attributes":{"velocity": {"type": "float", "
length": 8,"nullable" : true,"value": [120.0, 130.0, 140.0, 120.0]}, "
power": {"type": "float", "length": 4,"nullable" : true,"value": [120.
0, 130.0, 140.0, 120.0]}}}') a)
Select st_trajAttrsMeanMax(a, 'velocity') from traj;
 st_trajattrsmeanmax
---------------------
 ("@ 1 hour",135)
 ("@ 2 hours",130)
 ("@ 3 hours",127.5)
(3 rows)

 With traj AS (
    Select ST_makeTrajectory('STPOINT', 'LINESTRING(1 1, 6 6, 9 8, 10
12)'::geometry,
    ARRAY['2010-01-01 11:30'::timestamp, '2010-01-01 12:30', '2010-01-
01 13:30', '2010-01-01 14:30'],
    '{"leafcount":4, "attributes":{"velocity": {"type": "float", "
length": 8,"nullable" : true,"value": [120.0, 130.0, 140.0, 120.0]}, "
power": {"type": "float", "length": 4,"nullable" : true,"value": [120.
0, 130.0, 140.0, 120.0]}}}') a)
Select (st_trajAttrsMeanMax(a, 'velocity')).* from traj;
 duration |  max
----------+-------
 01:00:00 |   135
 02:00:00 |   130
 03:00:00 | 127.5
```

```
(3 rows)
```

6.7 空间关系判断

6.7.1 ST_intersects

指定时间区间的轨迹段和几何图形空间是否相交。

语法

```
boolean ST_intersects(trajectory traj, tsrange range, geometry g);
boolean ST_intersects(trajectory traj, timestamp t1, timestamp t2,
geometry g);
```

参数

参数名称	描述
traj	**轨迹对象。**
t1	**开始时间。**
t2	**结束时间。**
range	**时间段。**
g	**几何对象。**

示例

```
Select ST_intersects(traj, '2010-1-1 13:00:00', '2010-1-1 14:00:00', '
LINSTRING(0 0, 5 5, 9 9)'::geometry) from traj_table;
```

6.7.2 ST_equals

指定时间区间的轨迹段和几何图形空间是否相同。

语法

```
boolean ST_equals(trajectory traj, tsrange range, geometry g);
boolean ST_equals(trajectory traj, timestamp t1, timestamp t2,
geometry g);
```

参数

参数名称	描述
traj	**轨迹对象。**
t1	**开始时间。**
t2	**结束时间。**

参数名称	描述
range	时间段。
g	几何对象。

示例

```
Select ST_equals(traj, '2010-1-1 13:00:00', '2010-1-1 14:00:00', '
LINSTRING(0 0, 5 5, 9 9)'::geometry) from traj_table;
```

6.7.3 ST_distanceWithin

指定时间区间的轨迹段离给定几何对象在指定距离之内。

语法

```
boolean ST_distanceWithin(trajectory traj, tsrange range, geometry g,
float8 d);
boolean ST_distanceWithin(trajectory traj, timestamp t1, timestamp t2
, geometry g, float8 d);
```

参数

参数名称	描述
traj	轨迹对象。
t1	开始时间。
t2	结束时间。
range	时间段。
g	几何对象。
d	距离。

示例

```
Select ST_distanceWithin(traj, '2010-1-1 13:00:00', '2010-1-1 14:00:00
', 'LINSTRING(0 0, 5 5, 9 9)'::geometry, 10) from  traj_table;
```

6.8 空间处理

6.8.1 ST_intersection

给定时间段的轨迹和给定的几何对象执行相交处理，返回相交处理后的轨迹对象。

语法

```
trajectory[] ST_intersection(trajectory traj, tsrange range, geometry
g);
trajectory[] ST_intersection(trajectory traj, timestamp t1, timestamp
t2, geometry g);
```

参数

参数名称	描述
traj	**轨迹对象。**
t1	**开始时间。**
t2	**结束时间。**
range	**时间段。**
g	**几何对象。**

描述

如果轨迹几何对象和几何对象有多个交点，则返回多个子轨迹。

示例

```
Select ST_intersection(traj, '2010-1-1 13:00:00', '2010-1-1 14:00:00',
 'LINSTRING(0 0, 5 5, 9 9)'::geometry) from  traj_table;
```

6.8.2 ST_difference

给定时间段的轨迹和给定的几何对象执行相差处理，返回相交处理后的轨迹对象。

语法

```
trajectory ST_difference(trajectory traj, tsrange range, geometry g);
trajectory ST_difference(trajectory traj, timestamp t1, timestamp t2,
geometry g);
```

参数

参数名称	描述
traj	**轨迹对象。**
t1	**开始时间。**
t2	**结束时间。**

参数名称	描述
range	时间段。
g	几何对象。

示例

```
Select ST_difference(traj, '2010-1-1 13:00:00', '2010-1-1 14:00:00', '
LINSTRING(0 0, 5 5, 9 9)'::geometry) from  traj_table;
```

6.9 空间统计

6.9.1 ST_nearestApproachPoint

指定时间段的轨迹中找到与给定几何对象最邻近点信息。

语法

```
geometry ST_nearestApproachPoint(trajectory traj, tsrange range,
geometry g);
geometry ST_nearestApproachPoint(trajectory traj, timestamp t1,
timestamp t2, geometry g);
```

参数

参数名称	描述
traj	轨迹对象。
t1	开始时间。
t2	结束时间。
range	时间段。
g	几何对象。

示例

```
Select ST_nearestApproachPoint(traj, '2010-1-1 13:00:00', '2010-1-1 14
:00:00', 'LINSTRING(0 0, 5 5, 9 9)'::geometry) from  traj_table;
```

6.9.2 ST_nearestApproachDistance

指定时间段的轨迹中找到与给定几何对象最近距离。

语法

```
float8 ST_nearestApproachDistance(trajectory traj, tsrange range,
geometry g);
```

```
float8 ST_nearestApproachDistance(trajectory traj, timestamp t1,
timestamp t2, geometry g);
```

参数

参数名称	描述
traj	**轨迹对象。**
t1	**开始时间。**
t2	**结束时间。**
range	**时间段。**
g	**几何对象。**

示例

```
Select ST_nearestApproachDistance(traj, '2010-1-1 13:00:00', '2010-1-1
 14:00:00', 'LINSTRING(0 0, 5 5, 9 9)'::geometry) from  traj_table;
```

6.10 时空关系判断

6.10.1 ST_intersects

指定时间区间的轨迹段1和轨迹段2是否相交。

语法

```
boolean ST_intersects(trajectory traj1, trajectory traj2);
boolean ST_intersects(trajectory traj1, trajectory traj2, tsrange
range);
boolean ST_intersects(trajectory traj1, trajectory traj2, timestamp t1
, timestamp t2);
```

参数

参数名称	描述
traj	**轨迹对象。**
t1	**开始时间。**
t2	**结束时间。**
range	**时间段。**

示例

```
Select ST_intersects((Select traj from traj_table where id=1), (Select
 traj from traj_table where id=2), '2010-1-1 13:00:00', '2010-1-1 14:
00:00');
```

6.10.2 ST_equals

指定时间区间的轨迹段1和轨迹段2空间是否相等。

语法

```
boolean ST_equals(trajectory traj1, trajectory traj2, tsrange range);
boolean ST_equals(trajectory traj1, trajectory traj2, timestamp t1,
timestamp t2);
```

参数

参数名称	描述
traj	**轨迹对象。**
t1	**开始时间。**
t2	**结束时间。**
range	**时间段。**

示例

```
Select ST_equals((Select traj from traj_table where id=1), (Select
traj from traj_table where id=2), '2010-1-1 13:00:00', '2010-1-1 14:00
:00');
```

6.10.3 ST_distanceWithin

指定时间区间的轨迹段1离给定轨迹段2是否在指定距离之内。

语法

```
boolean ST_distanceWithin(trajectory traj1, trajectory traj2, tsrange
range, float8 d);
boolean ST_distanceWithin(trajectory traj1, trajectory traj2,
timestamp t1, timestamp t2, float8 d);
```

参数

参数名称	描述
traj	**轨迹对象。**
t1	**开始时间。**
t2	**结束时间。**

参数名称	描述
range	时间段。
d	指定距离。

示例

```
Select ST_distanceWithin((Select traj from traj_table where id=1), (
Select traj from traj_table where id=2), '2010-1-1 13:00:00', '2010-1-
1 14:00:00', 100);
```

6.10.4 ST_durationWithin

指定时间区间的轨迹段1和轨迹段2经过某点(空间相交点)的时间相差是否在指定时间区间内。

语法

```
boolean ST_durationWithin(trajectory traj1, trajectory traj2, tsrange
range, interval i );
boolean ST_durationWithin(trajectory traj1, trajectory traj2,
timestamp t1, timestamp t2, interval i);
```

参数

参数名称	描述
traj	轨迹对象。
t1	开始时间。
t2	结束时间。
range	时间段。
i	时间间隔。

描述

如果轨迹多次经过相同的点，任意一个时间符合要求就认为符合要求。

示例

```
Select ST_durationWithin((Select traj from traj_table where id=1), (
Select traj from traj_table where id=2), '2010-1-1 13:00:00', '2010-1-
1 14:00:00', INTERVAL '30s');
```

6.11 时空处理

6.11.1 ST_intersection

定时间段的轨迹1和轨迹2执行对应时间点上的移动对象之间的空间Intersection处理。

语法

```
geometry ST_intersection(trajectory traj1, trajectory traj2, tsrange
range);
geometry ST_intersection(trajectory traj1, trajectory traj2, timestamp
 t1, timestamp t2);
```

参数

参数名称	描述
traj1	**轨迹对象1。**
traj2	**轨迹对象2。**
t1	**开始时间。**
t2	**结束时间。**
range	**时间段。**

示例

```
Select ST_intersection((Select traj from traj_table where id=1), (
Select traj from traj_table where id=2), '2010-1-1 13:00:00', '2010-1-
1 14:00:00');
```

6.12 时空统计

6.12.1 ST_nearestApproachPoint

指定时间段的轨迹1和轨迹2中找到最邻近点信息。

语法

```
geometry ST_nearestApproachPoint(trajectory traj1, trajectory traj2);
geometry ST_nearestApproachPoint(trajectory traj1, trajectory traj2,
tsrange range);
geometry ST_nearestApproachPoint(trajectory traj1, trajectory traj2,
timestamp t1, timestamp t2);
```

参数

参数名称	描述
traj1	**轨迹对象1。**
traj2	**轨迹对象2。**

参数名称	描述
t1	开始时间。
t2	结束时间。
range	时间段。

示例

```
Select ST_nearestApproachPoint((Select traj from traj_table where id
=1), (Select traj from traj_table where id=2), '2010-1-1 13:00:00', '
2010-1-1 14:00:00');
```

6.12.2 ST_nearestApproachDistance

指定时间段的轨迹1和轨迹2中找到最邻近距离。

语法

```
float8 ST_nearestApproachDistance(trajectory traj,  trajectory traj2);
float8 ST_nearestApproachDistance(trajectory traj,  trajectory traj2,
tsrange range);
float8 ST_nearestApproachDistance(trajectory traj,  trajectory traj2,
timestamp t1, timestamp t2);
```

参数

参数名称	描述
traj1	轨迹对象1。
traj2	轨迹对象2。
t1	开始时间。
t2	结束时间。
range	时间段。

示例

```
Select ST_nearestApproachDistance((Select traj from traj_table where
id=1), (Select traj from traj_table where id=2), '2010-1-1 13:00:00',
 '2010-1-1 14:00:00');
```

6.13 距离测量

6.13.1 ST_length

获取一条轨迹的行程总长度，单位为米。

语法

```
float8 ST_length(trajectory traj, integer srid default 0);
```

参数

参数名称	描述
traj	轨迹对象。
sird	轨迹点坐标的空间参考值，默认为0（4326）。

描述

根据轨迹srid值计算椭球面长度，单位为米；通常trajectory对象中会有srid值，如果trajectory对象没有srid值，可通过函数参数srid指定，如果sird仍未知，函数将默认srid为4326。

示例

```
select st_length(traj) from traj where id = 2;
    st_length
------------------
 13494.6660605311
(1 row)
```

6.13.2 ST_euclideanDistance

计算两个轨迹对象之间的欧几里得距离。

语法

```
float ST_euclideanDistance(trajectory traj1, trajectory traj2);
```

参数

参数名称	描述
traj1	轨迹对象1。
traj2	轨迹对象2。

描述

距离已经进行了标准化处理。

示例

```
Select ST_euclideanDistance((Select traj from traj_table where id=1),
 (Select traj from traj_table where id=2));
```

6.13.3 ST_mdistance

计算轨迹对象中所有相同的时间点的欧几里得距离。

语法

```
float[] ST_mdistance(trajectory traj1, trajectory traj2);
```

参数

参数名称	描述
traj1	**轨迹对象1。**
traj2	**轨迹对象2。**

描述

未经过标准化处理。

示例

```
Select ST_mDistance((Select traj from traj_table where id=1), (Select
traj from traj_table where id=2));
```

6.14 相似度分析

6.14.1 ST_lcsSimilarity

计算基于LCSS（Longest Common Sub Sequence）算法的两条轨迹的相似度。

语法

```
integer ST_lcsSimilarity(trajectory traj1, trajectory traj2, float8
dist, distanceUnit unit default 'M' );
integer ST_lcsSimilarity(trajectory traj1, trajectory traj2, float8
dist, interval lag, distanceUnit unit default 'M');
```

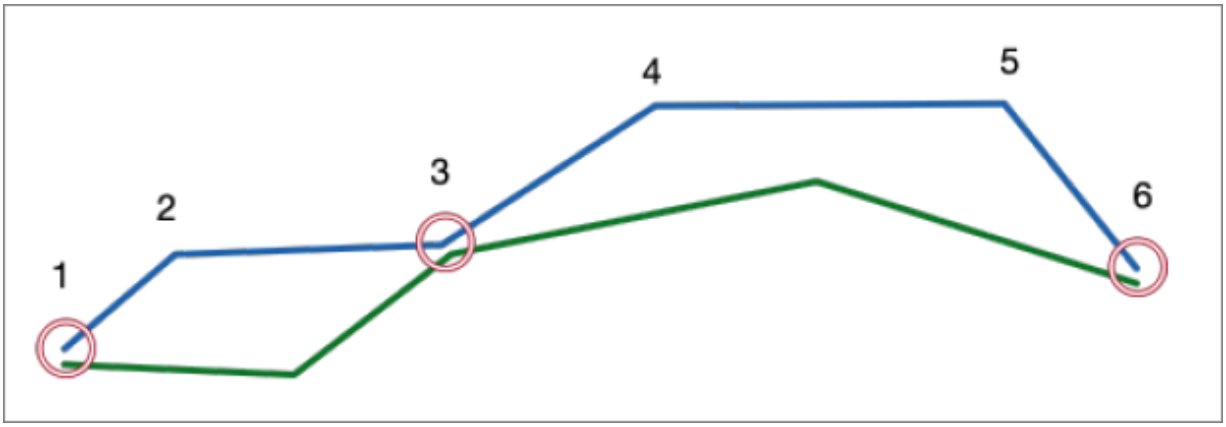
参数

参数名称	描述
traj1	**轨迹对象1。**
traj2	**轨迹对象2。**

参数名称	描述
dist	两点之间的距离容差，单位为米。
lag	两点之间的时间容差。
unit	距离单位，允许以下值： · 'M' : 米 · 'KM': 千米 · 'D': 度，只允许空间参考为 WGS84（4326）轨迹使用

描述

LCSS用于计算最大的公共子序列。用于判断两个轨迹点是否一致的条件包括空间距离和时间距离。 返回的结果是符合条件的轨迹点的数量 。



上图中轨迹点1，3， 6 符合要求，返回为 3。

通常trajectory对象中会有srid值，如果trajectory对象没有srid值，则默认为4326。

示例

```
With traj AS (
    Select ST_makeTrajectory('STPOINT', 'LINESTRINGZ(114.000528 33.
588163 54.87 , 114.000535 33.588235 54.85 , 114.000447 33.588272 54.
69 , 114.000348 33.588287 54.73 , 114.000245 33.588305 55.26 , 114.
000153 33.588305 55.3)'::geometry,
                            ARRAY['2010-01-01 11:30'::timestamp, '
2010-01-01 11:31', '2010-01-01 11:32', '2010-01-01 11:33','2010-01-01
11:34','2010-01-01 11:35'], NULL) a,
           ST_makeTrajectory('STPOINT', 'LINESTRINGZ(114.000529 33.
588163 54.87 , 114.000535 33.578235 54.85 , 114.000447 33.578272 54.
69 , 114.000348 33.578287 54.73 , 114.000245 33.578305 55.26 , 114.
000163 33.588305 55.3)'::geometry,
                            ARRAY['2010-01-01 11:29:58'::timestamp, '
2010-01-01 11:31:02', '2010-01-01 11:33', '2010-01-01 11:33:09','2010-
01-01 11:34','2010-01-01 11:34:30'], NULL) b)
Select  st_LCSSimilarity(a, b, 100) from traj;
  st_lcssimilarity
-------------------
 2
```

```
 (1 row)

 With traj AS (
    Select ST_makeTrajectory('STPOINT', 'LINESTRINGZ(114.000528 33.
588163 54.87 , 114.000535 33.588235 54.85 , 114.000447 33.588272 54.
69 , 114.000348 33.588287 54.73 , 114.000245 33.588305 55.26 , 114.
000153 33.588305 55.3)'::geometry,
                            ARRAY['2010-01-01 11:30'::timestamp, '
2010-01-01 11:31', '2010-01-01 11:32', '2010-01-01 11:33','2010-01-01
11:34','2010-01-01 11:35'], NULL) a,
           ST_makeTrajectory('STPOINT', 'LINESTRINGZ(114.000529 33.
588163 54.87 , 114.000535 33.578235 54.85 , 114.000447 33.578272 54.
69 , 114.000348 33.578287 54.73 , 114.000245 33.578305 55.26 , 114.
000163 33.588305 55.3)'::geometry,
                            ARRAY['2010-01-01 11:29:58'::timestamp, '
2010-01-01 11:31:02', '2010-01-01 11:33', '2010-01-01 11:34:15','2010-
01-01 11:34:50','2010-01-01 11:34:30'], NULL) b)
Select st_LCSSimilarity(a, b, 100, interval '30 seconds') from traj;
 st_lcssimilarity
------------------
2
(1 row)
```

6.14.2 ST_lcsDistance

计算基于LCSS（Longest Common Sub Sequence）算法的两条轨迹的距离。

语法

```
float8 ST_lcsDisatance(trajectory traj1, trajectory traj2, float8 dist
, distanceUnit unit default 'M');
float8 ST_lcsDisatance(trajectory traj1, trajectory traj2, float8 dist
, interval lag, distanceUnit unit default 'M');
```

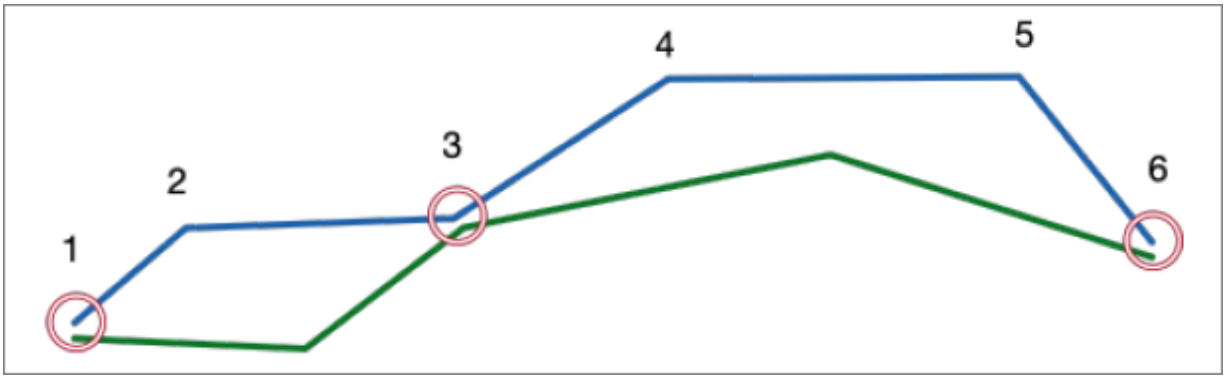
参数

参数名称	描述
traj1	**轨迹对象1。**
traj2	**轨迹对象2。**
dist	**两点之间的距离容差，单位为米。**
lag	**两点之间的时间容差。**
unit	**距离单位，允许以下值：** · **'M' : 米** · **'KM': 千米** · **'D': 度，只允许空间参考为 WGS84（4326）轨迹使用**

描述

LCSS用于计算最大的公共子序列。用于判断两个轨迹点是否一致的条件包括空间距离和时间距离。

返回的结果是 1- （LCSS）/ min(leafcount(traj1), leafcount(traj2)。



上图中轨迹点1，3，6符合要求，LCSS的数量为3， 结果为1-3/5=0.4。

通常trajectory对象中会有srid值，如果trajectory对象没有srid值，则默认为4326。

示例

```
With traj AS (
    Select ST_makeTrajectory('STPOINT', 'LINESTRINGZ(114.000528 33.
588163 54.87 , 114.000535 33.588235 54.85 , 114.000447 33.588272 54.
69 , 114.000348 33.588287 54.73 , 114.000245 33.588305 55.26 , 114.
000153 33.588305 55.3)'::geometry,
                              ARRAY['2010-01-01 11:30'::timestamp, '
2010-01-01 11:31', '2010-01-01 11:32', '2010-01-01 11:33','2010-01-01
11:34','2010-01-01 11:35'], NULL) a,
           ST_makeTrajectory('STPOINT', 'LINESTRINGZ(114.000529 33.
588163 54.87 , 114.000535 33.578235 54.85 , 114.000447 33.578272 54.
69 , 114.000348 33.578287 54.73 , 114.000245 33.578305 55.26 , 114.
000163 33.588305 55.3)'::geometry,
                              ARRAY['2010-01-01 11:29:58'::timestamp, '
2010-01-01 11:31:02', '2010-01-01 11:33', '2010-01-01 11:33:09','2010-
01-01 11:34','2010-01-01 11:34:30'], NULL) b)
Select  st_LCSDistance(a, b, 100) from traj;
  st_lcsdistance
------------------
 0.666666666666667
 (1 row)

 With traj AS (
    Select ST_makeTrajectory('STPOINT', 'LINESTRINGZ(114.000528 33.
588163 54.87 , 114.000535 33.588235 54.85 , 114.000447 33.588272 54.
69 , 114.000348 33.588287 54.73 , 114.000245 33.588305 55.26 , 114.
000153 33.588305 55.3)'::geometry,
                              ARRAY['2010-01-01 11:30'::timestamp, '
2010-01-01 11:31', '2010-01-01 11:32', '2010-01-01 11:33','2010-01-01
11:34','2010-01-01 11:35'], NULL) a,
           ST_makeTrajectory('STPOINT', 'LINESTRINGZ(114.000529 33.
588163 54.87 , 114.000535 33.578235 54.85 , 114.000447 33.578272 54.
69 , 114.000348 33.578287 54.73 , 114.000245 33.578305 55.26 , 114.
000163 33.588305 55.3)'::geometry,
                              ARRAY['2010-01-01 11:29:58'::timestamp, '
2010-01-01 11:31:02', '2010-01-01 11:33', '2010-01-01 11:34:15','2010-
01-01 11:34:50','2010-01-01 11:34:30'], NULL) b)
Select st_LCSDistance(a, b, 100, interval '30 seconds') from traj;
 st_lcsdistance
------------------
0.666666666666667
```

```
(1 row)

With traj AS (
    Select ST_makeTrajectory('STPOINT', 'LINESTRINGZ(114.000528 33.
588163 54.87 , 114.000535 33.588235 54.85 , 114.000447 33.588272 54.
69 , 114.000348 33.588287 54.73 , 114.000245 33.588305 55.26 , 114.
000153 33.588305 55.3)'::geometry,
                            ARRAY['2010-01-01 11:30'::timestamp, '
2010-01-01 11:31', '2010-01-01 11:32', '2010-01-01 11:33','2010-01-01
11:34','2010-01-01 11:35'], NULL) a,
           ST_makeTrajectory('STPOINT', 'LINESTRINGZ(114.000529 33.
588163 54.87 , 114.000535 33.578235 54.85 , 114.000447 33.578272 54.
69 , 114.000348 33.578287 54.73 , 114.000245 33.578305 55.26 , 114.
000163 33.588305 55.3)'::geometry,
                            ARRAY['2010-01-01 11:29:58'::timestamp, '
2010-01-01 11:31:02', '2010-01-01 11:33', '2010-01-01 11:34:15','2010-
01-01 11:34:50','2010-01-01 11:34:30'], NULL) b)
Select st_LCSDistance(a, b, 100, interval '30 seconds', 'M') from traj
;
 st_lcsdistance
--------------------
0.666666666666667
(1 row)
```

6.14.3 ST_lcsSubDistance

计算LCSS轨迹段与traj1在这段轨迹上的距离。

语法

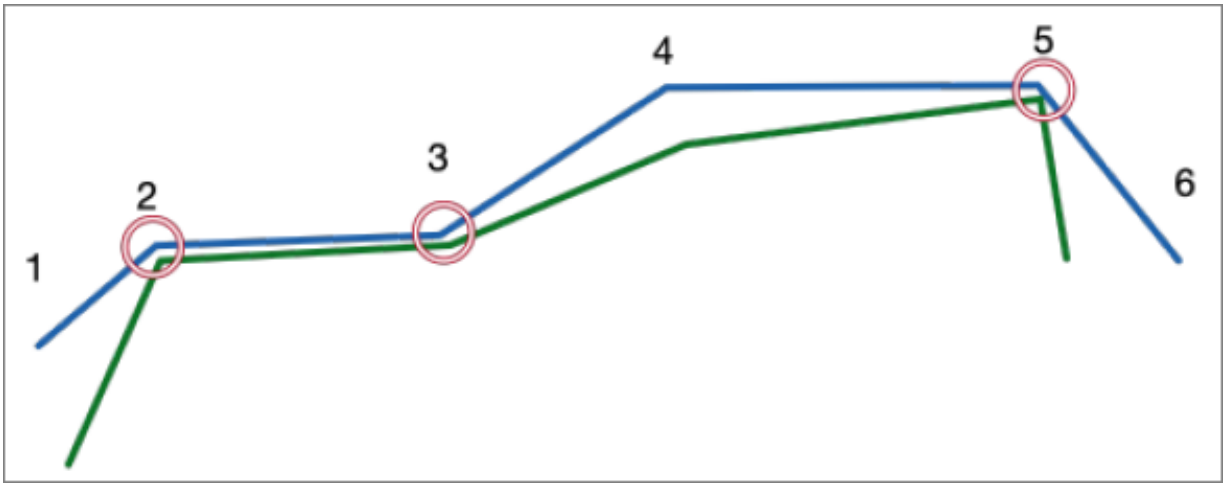
```
float8 ST_lcsSubDisatance(trajectory traj1, trajectory traj2, float8
dist, distanceUnit unit default 'M');
float8 ST_lcsSubDisatance(trajectory traj1, trajectory traj2, float8
dist, interval lag, distanceUnit unit default 'M');
```

参数

参数名称	描述
traj1	**轨迹对象1。**
traj2	**轨迹对象2。**
dist	**两点之间的距离容差，单位为米。**
lag	**两点之间的时间容差。**
unit	**距离单位，允许以下值：** · **'M' : 米** · **'KM': 千米** · **'D': 度，只允许空间参考为 WGS84（4326）轨迹使用**

描述

本函数计算的是与LCSS轨迹段相对应的轨迹1子轨迹的点数与LCSS轨迹段的点数的比例关系。



上图中轨迹点[2,3,5]为LCSS轨迹段，轨迹1与此对应的轨迹段为[2,3,4,5]，返回的结果为1-3/4。

数值越小表明LCSS轨迹与轨迹1的相似度越高。

通常trajectory对象中会有srid值，如果trajectory对象没有srid值，则默认为4326。

示例

```
With traj AS (
    Select ST_makeTrajectory('STPOINT', 'LINESTRINGZ(114.000528 33.
588163 54.87 , 114.000535 33.588235 54.85 , 114.000447 33.588272 54.
69 , 114.000348 33.588287 54.73 , 114.000245 33.588305 55.26 , 114.
000153 33.588305 55.3)'::geometry,
                            ARRAY['2010-01-01 11:30'::timestamp, '
2010-01-01 11:31', '2010-01-01 11:32', '2010-01-01 11:33','2010-01-01
11:34','2010-01-01 11:35'], NULL) a,
           ST_makeTrajectory('STPOINT', 'LINESTRINGZ(114.000529 33.
588163 54.87 , 114.000535 33.578235 54.85 , 114.000447 33.578272 54.
69 , 114.000348 33.578287 54.73 , 114.000245 33.578305 55.26 , 114.
000163 33.588305 55.3)'::geometry,
                            ARRAY['2010-01-01 11:29:58'::timestamp, '
2010-01-01 11:31:02', '2010-01-01 11:33', '2010-01-01 11:33:09','2010-
01-01 11:34','2010-01-01 11:34:30'], NULL) b)
Select st_LCSSubDistance(a, b, 100) from traj;

st_lcssubdistance
------------------------
 0.6666666666666666667
 (1 row)

 With traj AS (
    Select ST_makeTrajectory('STPOINT', 'LINESTRINGZ(114.000528 33.
588163 54.87 , 114.000535 33.588235 54.85 , 114.000447 33.588272 54.
69 , 114.000348 33.588287 54.73 , 114.000245 33.588305 55.26 , 114.
000153 33.588305 55.3)'::geometry,
                            ARRAY['2010-01-01 11:30'::timestamp, '
2010-01-01 11:31', '2010-01-01 11:32', '2010-01-01 11:33','2010-01-01
11:34','2010-01-01 11:35'], NULL) a,
           ST_makeTrajectory('STPOINT', 'LINESTRINGZ(114.000529 33.
588163 54.87 , 114.000535 33.578235 54.85 , 114.000447 33.578272 54.
69 , 114.000348 33.578287 54.73 , 114.000245 33.578305 55.26 , 114.
000163 33.588305 55.3)'::geometry,
```

```
                              ARRAY['2010-01-01 11:29:58'::timestamp, '
2010-01-01 11:31:02', '2010-01-01 11:33', '2010-01-01 11:33:09','2010-
01-01 11:34','2010-01-01 11:34:30'], NULL) b)
Select st_LCSSubDistance(a, b, 100, interval '30 seconds') from traj;
st_lcssubdistance
-------------------
0.666666666666667
(1 row)
```

6.15 变量

6.15.1 ganos.trajectory.attr_string_length

设置字符串类型属性默认长度。

描述

`attr_string_length`**为integer类型。**

示例

```
Set ganos.trajectory.attr_string_length = 32;
```

7 Trajectory 最佳实践

使用合适的时空索引

合适的索引能加快查询速度，需要根据应用场景的要求来构建合适的索引。可针对轨迹数据类型构建以下索引：

- 空间索引：只针对轨迹的空间范围建立索引，适合只查询轨迹空间范围情况。
- 时间索引：只针对轨迹的时间范围建立索引，适合只查询轨迹时间范围情况。
- 时空复合索引：建立时空联合索引，适合时间和空间同时过滤查询。

```
--创建基于函数的空间索引，加速空间过滤
create index tr_spatial_geometry_index on trajtab using gist (
st_trajectoryspatial(traj));


--创建基于函数的时间段索引，加速时间过滤
create index tr_timespan_time_index on trajtab using gist (st_timespan
(traj));

--创建基于函数的轨迹起止时间
create index tr_starttime_index on trajtab using btree (st_starttime(
traj));
create index tr_endtime_index on trajtab using btree (st_endtime(traj
));


--首先创建btree_gist扩展
create extension btree_gist;

--建立btree_gist 起始时间、终止时间、空间复合索引
create index tr_traj_test_stm_etm_sp_index on traj_test using gist (
st_starttime(traj),st_endtime(traj),st_trajectoryspatial(traj));
```

采用合理的分区表

随着使用时间的增加，数据库中的轨迹数据量也不断增加，导致数据库索引变大，查询变慢。您可考虑采用分区表的模式降低单表数据量。

使用分区表请参考PostgreSQL文档中分区表相关章节。

减少使用字符串类型属性

轨迹属性中如有大量的字符串属性，会导致存储空间浪费和性能下降。

- 如果字符串为固定内容，可以转换为整型进行枚举，建议在程序中进行转换；
- 如果无法避免使用字符串类型属性，可指定字符串类型默认长度，避免空间浪费。

  设置字符串类型默认长度方法如下：

  ```
  -- 设置字符串类型默认长度为32
  ```

```
Set ganos.trajectory.attr_string_length = 32;
```

采用批量轨迹生成模式

使用批量轨迹点生成轨迹模式，避免采取单个轨迹点追加模式。

采用优化压缩模式

lz4 压缩算法是一种优秀的压缩算法，具有更高的压缩率和更快的执行速度。如果要启用lz4压缩算法，设置方法如下：

```
-- 设置使用lz4 压缩
Set toast_compression_use_lz4 = true;

-- 使用pg默认压缩算法
Set toast_compression_use_lz4 = false;
```

如果要对整个数据默认采用lz4压缩算法，设置方法如下：

```
-- 数据库使用lz4压缩
Alter database dbname Set toast_compression_use_lz4 = true;

-- 数据库使用默认压缩
Alter database dbname Set toast_compression_use_lz4 = false;
```

8 Trajecotry 常见问题

如何把坐标点数据转换为轨迹对象?

采用 ST_MakeTrajectory 构造函数可以将其转换为轨迹对象，设置方法如下:

```
-- 创建扩展
create extension ganos_trajectory cascade;

-- 创建点表
create table points(id integer, x float8, y float8, t timestamp, speed
 float8);
insert into points values(1, 128.1, 28.1, '2019-01-01 00:00:00', 100);
insert into points values(2, 128.2, 28.2, '2019-01-01 00:00:01', 101);
insert into points values(3, 128.3, 28.3, '2019-01-01 00:00:02', 102);
insert into points values(4, 128.4, 28.4, '2019-01-01 00:00:04', 103);

-- 创建轨迹表
create table traj(id integer, traj trajectory);

-- 插入数据
insert into traj(id, traj)
select 1,
    ST_MakeTrajectory('STPOINT'::leaftype, x, y, 4326, t, ARRAY['speed
'], NULL, s, NULL)
FROM (select array_agg(x order by id) as x,
      array_agg(y order by id) as y,
      array_agg(t order by id) as t,
      array_agg(speed order by id) as s
  From points) a;
```

系统自带的ST_MakTrajectory构造函数不满足需求怎么办?

如果系统自带的ST_MakeTrajectory 函数参数不符合应用需求，您可以进行自定义扩展。例如轨迹对象包含2个int8，2个float4和1个timestamp类型的属性，可以创建如下构造函数:

```
CREATE OR REPLACE FUNCTION  ST_MakeTrajectory(type leaftype, x float8
[], y float8[] ,
                srid integer, timespan timestamp[],attrs_name cstring
[], attr1 int8[],
                attr2 int8[], attr3 float4[], attr4 float4[], attr5
timestamp[])
    RETURNS trajectory
    AS '$libdir/libpg-trajectory16','sqltr_traj_make_all_array'
    LANGUAGE 'c' IMMUTABLE Parallel SAFE;
```

其中前6个参数为固定类型的参数，后5个参数为用户自定义的轨迹属性类型。使用时直接使用用户定义的重载函数即可。

如何向轨迹中追加轨迹点?

使用 ST_append 函数向现有轨迹追加轨迹点，ST_append 函数声明如下:

```
trajectory ST_append(trajectory traj, geometry spatial, timestamp[]
timespan, text str_theme_json) ;
```

```
trajectory ST_append(trajectory traj, trajectroy tail) ;
```

例如，基于点表向轨迹中追加轨迹点方法如下：

```
-- 创建扩展
create extension ganos_trajectory cascade;

-- 创建点表
create table points(id integer, x float8, y float8, t timestamp, speed
 float8);
insert into points values(1, 128.1, 28.1, '2019-01-01 00:00:00', 100);
insert into points values(2, 128.2, 28.2, '2019-01-01 00:00:01', 101);
insert into points values(3, 128.3, 28.3, '2019-01-01 00:00:02', 102);
insert into points values(4, 128.4, 28.4, '2019-01-01 00:00:04', 103);

-- 创建轨迹表
create table traj(id integer, traj trajectory);

-- 插入数据
insert into traj(id, traj)
select 1,
    ST_MakeTrajectory('STPOINT'::leaftype, x, y, 4326, t, ARRAY['speed
'], NULL, s, NULL)
FROM (select array_agg(x order by id) as x,
      array_agg(y order by id) as y,
      array_agg(t order by id) as t,
      array_agg(speed order by id) as s
  From points) a;

-- 插入新轨迹点
insert into points values(5, 128.5, 28.5, '2019-01-01 00:00:05', 105);
insert into points values(6, 128.6, 28.6, '2019-01-01 00:00:06', 106);
insert into points values(7, 128.7, 28.7, '2019-01-01 00:00:07', 107);

-- 基于单点更新轨迹
With point_traj as (
 select ST_MakeTrajectory('STPOINT'::leaftype, x, y, 4326, t, ARRAY['
speed'], NULL, s, NULL) AS traj
 FROM (select array_agg(x order by id) as x,
      array_agg(y order by id) as y,
      array_agg(t order by id) as t,
      array_agg(speed order by id) as s
      From points WHERE ID = 5) a
)
Update traj
set traj = ST_append(traj.traj, a.traj)
From point_traj a
WHERE traj.ID=1;

-- 如果使用程序生成sql也可以写成
With point_traj as (
 select ST_MakeTrajectory('STPOINT'::leaftype, ARRAY[128.5::float8],
      ARRAY[28.5::float8], 4326, ARRAY['2019-01-01 00:00:05'::
timestamp],
      ARRAY['speed'], NULL, ARRAY[106::float8], NULL) AS traj
)
Update traj
set traj = ST_append(traj.traj, a.traj)
From point_traj a
WHERE traj.ID =1;

-- 基于多点更新轨迹
```

```
With point_traj as (
 select ST_MakeTrajectory('STPOINT'::leaftype, x, y, 4326, t, ARRAY['
speed'], NULL, s, NULL) AS traj
 FROM (select array_agg(x order by id) as x,
      array_agg(y order by id) as y,
      array_agg(t order by id) as t,
      array_agg(speed order by id) as s
      From points WHERE ID > 5 ) a
)
Update traj
set traj = ST_append(traj.traj, a.traj)
From point_traj a
WHERE traj.ID =1;
```

如何启用优化压缩功能?

lz4 压缩算法是一种优秀的压缩算法，具有更高的压缩率和更快的执行速度。如果要启用lz4压缩算法，设置方法如下：

```
-- 设置使用lz4 压缩
Set toast_compression_use_lz4 = true;

-- 使用pg默认压缩算法
Set toast_compression_use_lz4 = false;
```

如果要对整个数据默认采用lz4压缩算法，设置方法如下：

```
-- 数据库使用lz4压缩
Alter database dbname Set toast_compression_use_lz4 = true;

-- 数据库使用默认压缩
Alter database dbname Set toast_compression_use_lz4 = false;
```

如何设置字符串类型属性默认长度?

guc变量ganos.trajectory.attr_string_length用于设置字符串类型属性。设置方法如下：

```
Set ganos.trajectory.attr_string_length = 32;
```

如何计算某个属性的最大值（最小值/平均值）？

计算某个属性的最大值（最小值/平均值），方法如下：

```
With traj AS (
  select '{"trajectory":{"version":1,"type":"STPOINT","leafcount":2,"
start_time":"2010-01-01 11:30:00","end_time":"2010-01-01 12:30:00","
spatial":"SRID=4326;LINESTRING(1 1,3 5)","timeline":["2010-01-01 11:30
:00","2010-01-01 12:30:00"],"attributes":{"leafcount":2,"velocity":{"
type":"integer","length":4,"nullable":true,"value":[1,100]},"speed":{"
type":"float","length":8,"nullable":true,"value":[null,1.0]},"angel":
{"type":"string","length":64,"nullable":true,"value":["test",null]}, "
tngel2":{"type":"timestamp","length":8,"nullable":true,"value":["2010-
01-01 12:30:00",null]},"bearing":{"type":"bool","length":1,"nullable":
true,"value":[null,true]}}}}'::trajectory a)
select avg(v) from
(
Select unnest(st_trajAttrsAsInteger(a, 'velocity')) as v from traj
```

```
) t;
```

9 服务发布

9.1 GeoServer

9.1.1 GeoServer简介

GeoServer是OpenGIS Web服务器规范的J2EE实现，利用GeoServer可以方便的发布地图数据，允许用户对特征数据进行更新、删除、插入操作，通过GeoServer可以在用户之间迅速共享空间地理信息。GeoServer兼容WMS和WFS两种OGC规范特性，支持PostgreSQL、Shapefile、ArcSDE、Oracle、VPF、MySQL、MapInfo，并支持上百种投影；同时能够将网络地图输出为jpeg、gif、png 、SVG 、KML等格式。GeoServer能够运行在任何基于J2EE/Servlet容器之上，嵌入MapBuilder支持AJAX的地图客户端OpenLayers，除此之外还包括许多其他的特性。

安装包下载

Tomcat+Geoserver+RestAPI完整安装包

9.1.2 发布几何数据

通过GeoServer发布Ganos中的几何数据。

添加数据源

1. 打开GeoServer，选择数据存储。

2. 单击添加新的数据存储，选择PostGIS。

3. 填写Ganos数据库的连接信息。


新建矢量数据源
添加一个新的矢量数据源
PostGIS
PostGIS Database
存储库的基本信息
工作区 *
poc
数据源名称 *
beijing
说明
beijing geometries
启用
连接参数
host *
pgm-2ze2m84 aliyuncs.com
port *
3432
database
ganos
schema
public
user *
ganos_test
passwd
••••••••••
命名空间 *

REST API

几何数据发布后，GeoServer提供给客户调用、访问的REST API，请参见 官网文档

9.1.3 发布栅格数据

通过GeoServer发布Ganos中的栅格数据。

添加数据源

1. 打开GeoServer，选择数据存储。

2. 单击添加新的数据存储，选择GanosRaster(PG/PolarDB)。

3. 填写Ganos数据库的连接信息。

参数说明如下。

参数名称	描述	示例
host	数据库地址（IP或RDS连接地址）。	xxxxxxx.pg.rds.aliyuncs.com
port	数据库端口号。	3432
database	数据库名称。	rasterdb
username	数据库用户名。	pguser
password	数据库密码。	123456
schema	表所在的schema。	默认为public

参数名称	描述	示例
table	栅格所在表名。	raster_table
column name	栅格列名称。	raster_column
filter	栅格对象过滤条件，为SQL 语句中的where条件。 如果符合条件的栅格有多个，总是使用第一个。	id=1或name='srtm'
name	geoserver显示的栅格名称。	myraster

REST API

- 接口：http://host:port/geoserver/rest/workspaces/{workspace}/coveragestores
- 方法：POST
- 参数：
  - workspace：已经创建好的workspace名称。
  - datastore body：示例如下，其中type为固定写法，url中填写阿里云pg/polardb的连接参数（json格式）。

```
{
  "coverageStore": {
    "name": "<datasource_name>",
    "type": "GanosRaster(PG/PolarDB)",
    "enabled":"true",
    "workspace":"<wokrspace>",
    "url": "{\"column\":\"<raster_colunm>\",\"database\":\"<
database_name>\",\"filter\":\"<raster_filter>\",\"host\":\"<
pg_host>\",\"name\":\"<public_name>\",\"password\":\"<user_passw
ord>\",\"port\":<pg_port>,\"schema\":\"<schema_name>\",\"ssl\":
false,\"table\":\"<raster_table_name>\",\"userName\":\"<user_name>
\",\"valid\":true}"
  }
}
```

创建数据源示例

```
{
  "coverageStore": {
    "name": "srtm",
    "type": "GanosRaster(PG/PolarDB)",
    "enabled":"true",
    "workspace":"test",
    "url": "{\"column\":\"rast\",\"database\":\"test_db\",\"filter
\":\"name='srtm'\",\"host\":\"pgm-xxxxxxxx.pg.rds.aliyuncs.com\",\"
name\":\"srtm_image\",\"password\":\"xxxx\",\"port\":3432,\"schema
\":\"public\",\"ssl\":true,\"table\":\"raster_table\",\"userName\":\"
raster_user\",\"valid\":true}"
  }
}
```

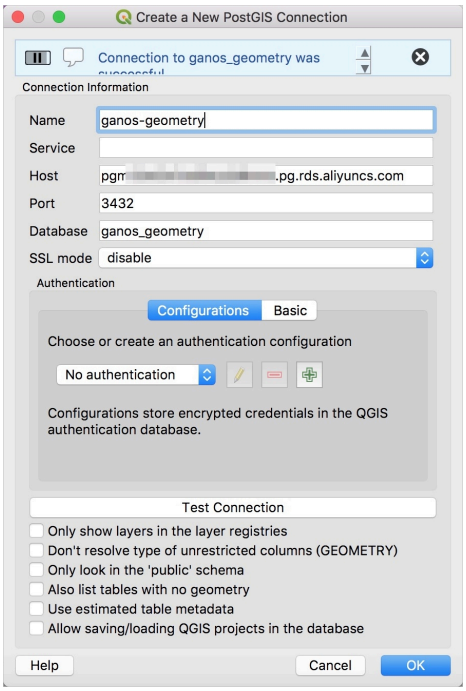
10 桌面应用

10.1 QGIS访问Ganos

QGIS（原称Quantum GIS）是一个用户界面友好的开源桌面端软件，支持多种矢量、栅格格式以及数据库作为数据源，同时支持数据的可视化、管理、编辑、分析以及印刷地图的制作。

创建Ganos连接

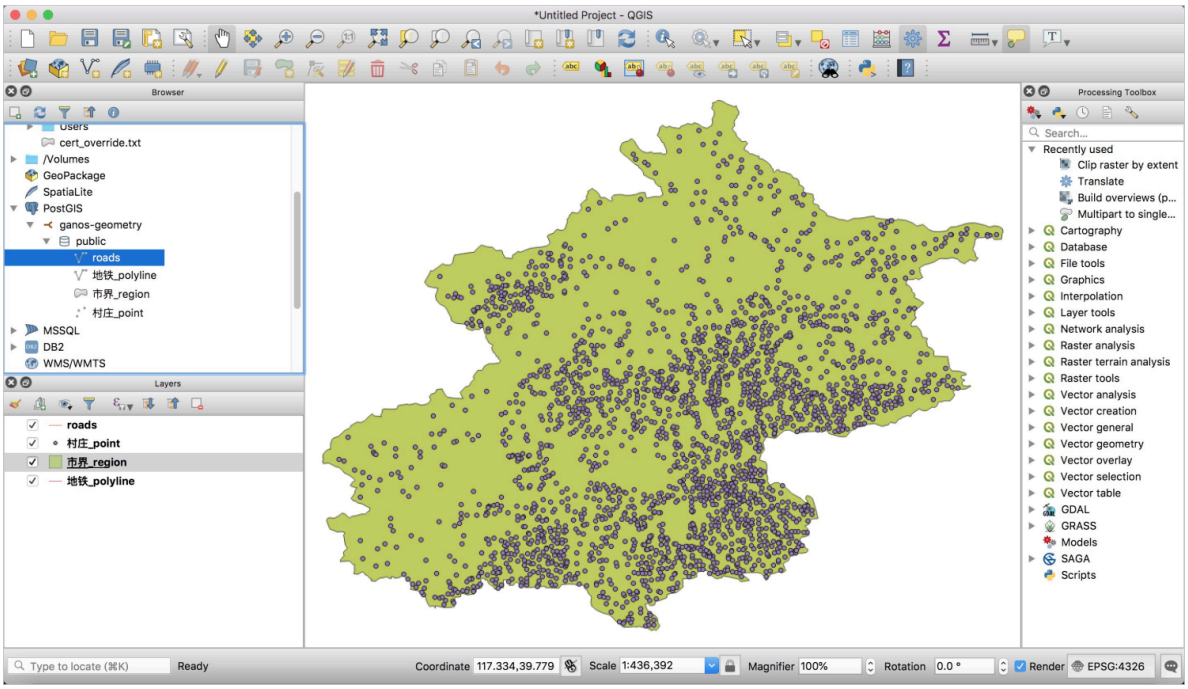
1. 直接创建PostGIS Connection，填入必要项参数，点击Test Connection。



参数说明如下。

参数	说明
Name	自定义连接名。
Host	数据库实例的IP地址或RDS/POLARDB外网地址，可以从控制台中获得。
Port	数据库实例的端口地址，可以从控制台中获得。
Database	需要连接的数据库名。
SSL Mode	SSL加密状态，选择Disable。

2. 找到建立的Ganos数据源，显示出已有的数据库实例以及实例下的空间图层，双击指定的空间图层，即可浏览、查看、编辑Ganos中的空间数据。

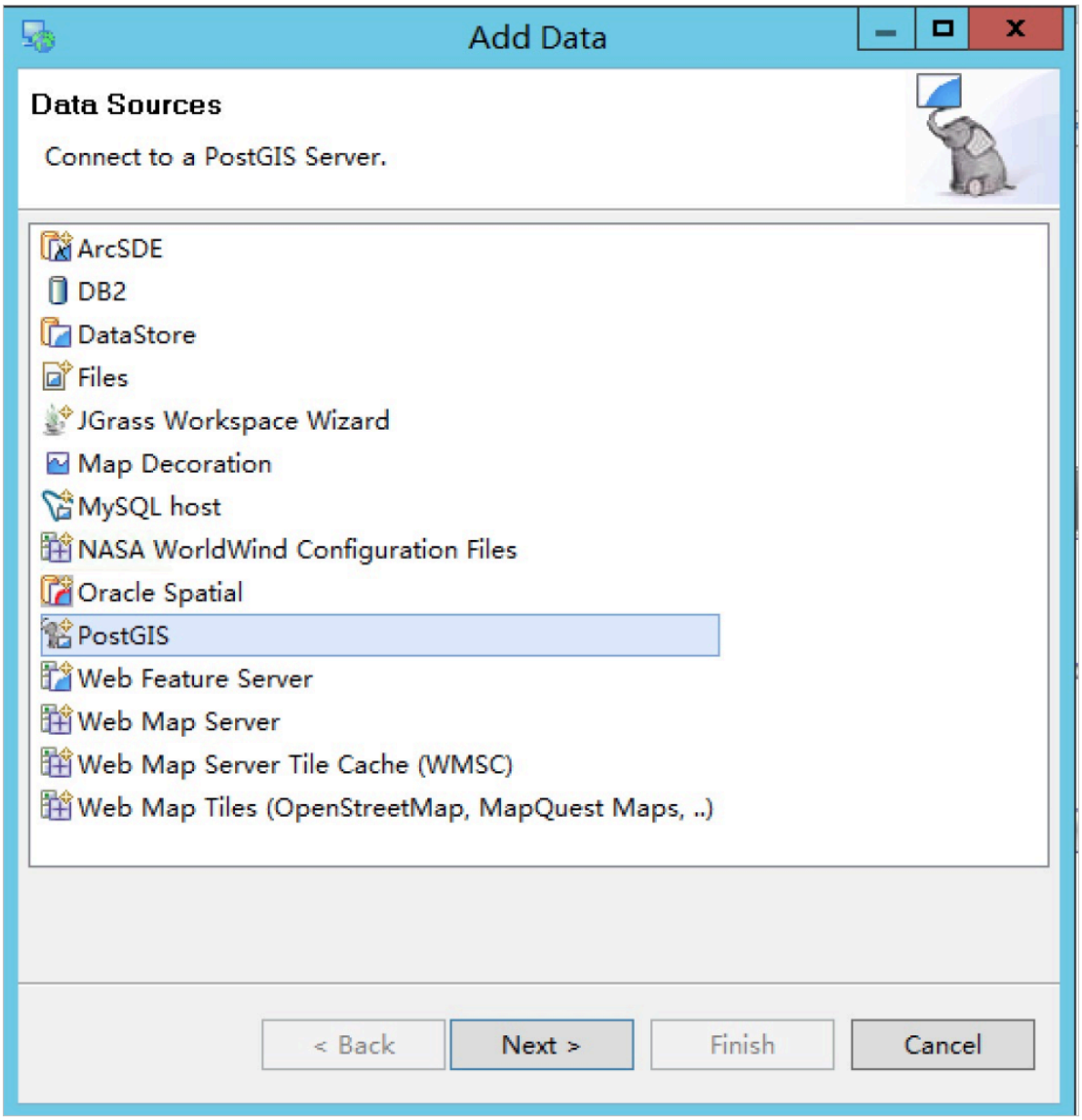


10.2 uDig访问Ganos

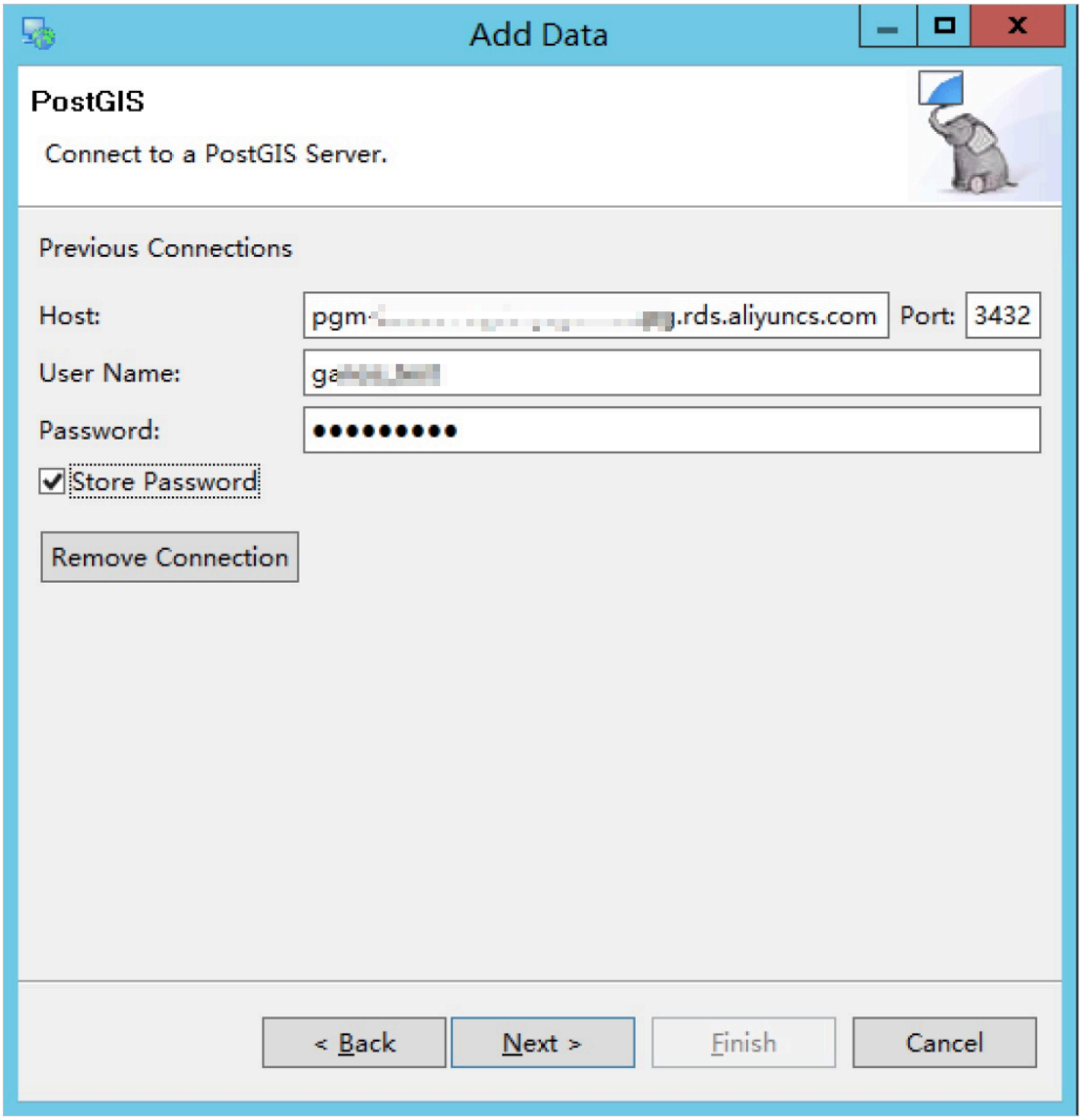
uDig是一款开源的桌面GIS应用和开发框架，可以进行空间数据（如shp地图文件）的编辑和查看，支持OpenGIS标准，对互联网GIS、网络地图服务器和网络功能服务器有特别的加强。uDig提供了一个通用的java平台来利用开源组件建设空间应用程序。

创建Ganos连接

1. 在软件界面选择Layer > Add Data。



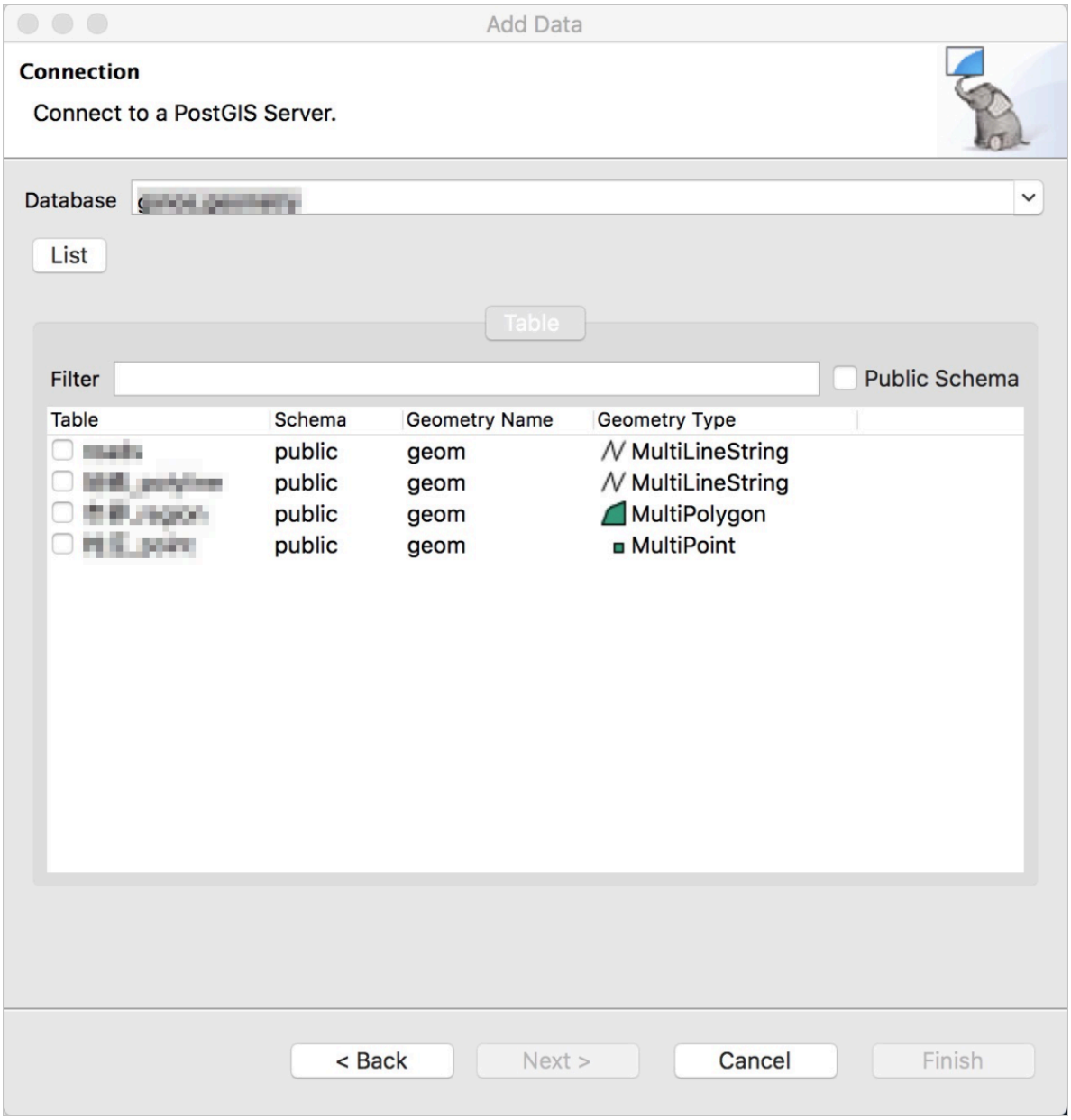
2. 填入Ganos数据库连接信息。



参数说明如下。

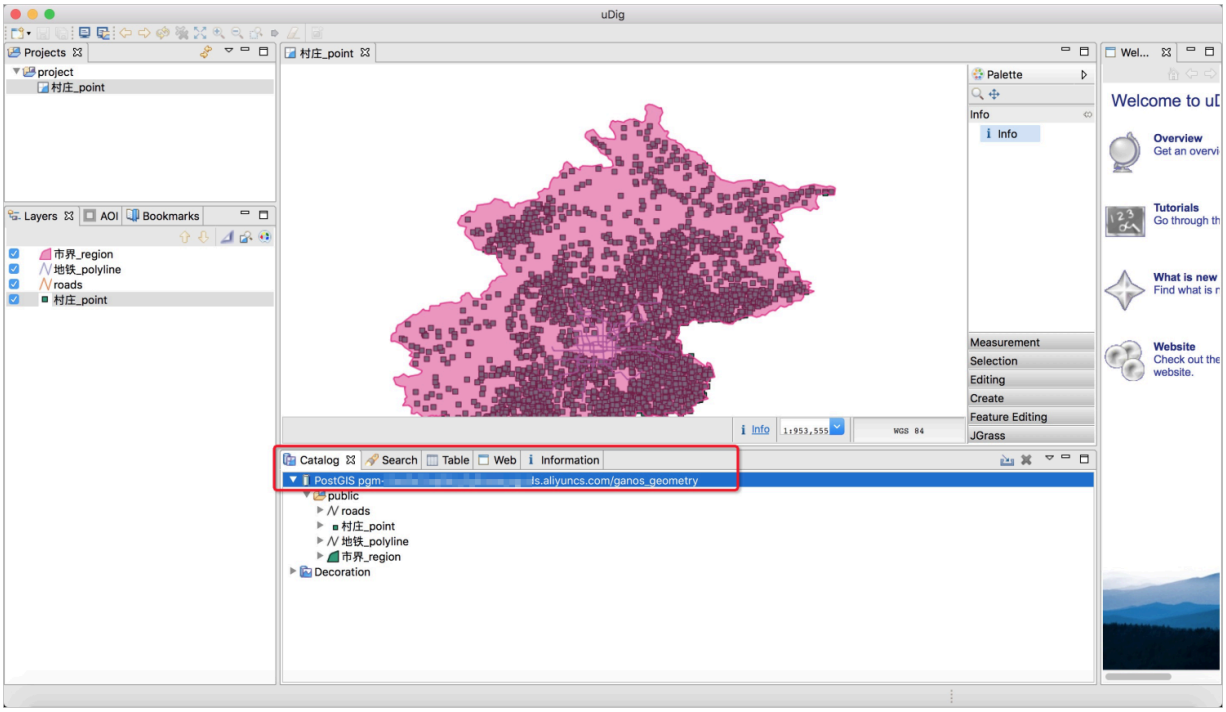
参数	说明
Host	数据库实例的IP地址或RDS/POLARDB外网地址，可以从控制台中获得。
Port	数据库实例的端口地址，可以从控制台中获得。
User Name	数据库用户名。
Password	密码。

3. 选择数据库实例。



数据浏览与操作

连接Ganos后，可以对数据库中对图层进行浏览、查看、编辑Ganos中的空间数据。



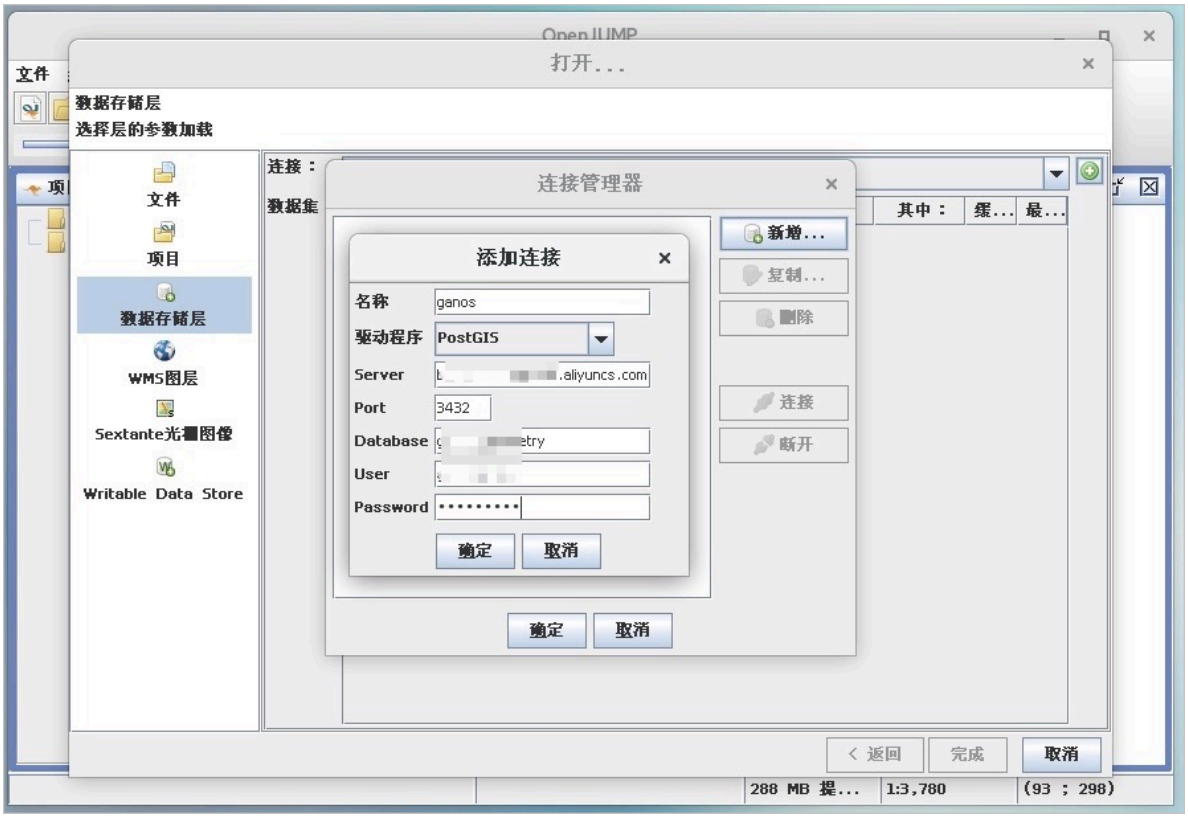
10.3 OpenJump访问Ganos

OpenJUMP是一套基于Java桌面GIS应用程序，其内置了地图可视化功能，可以用来实现特定的空间分析计算，包括Buffer缓冲区分析、Intersection叠加求交、Union叠加求和等空间分析功能，并可以通过插件方式为OpenJUMP进行功能的定制或拓展。

创建Ganos连接

1. **在软件界面选择文件 > 打开，选择数据存储层。**
2. **在连接管理器中选择新增。**

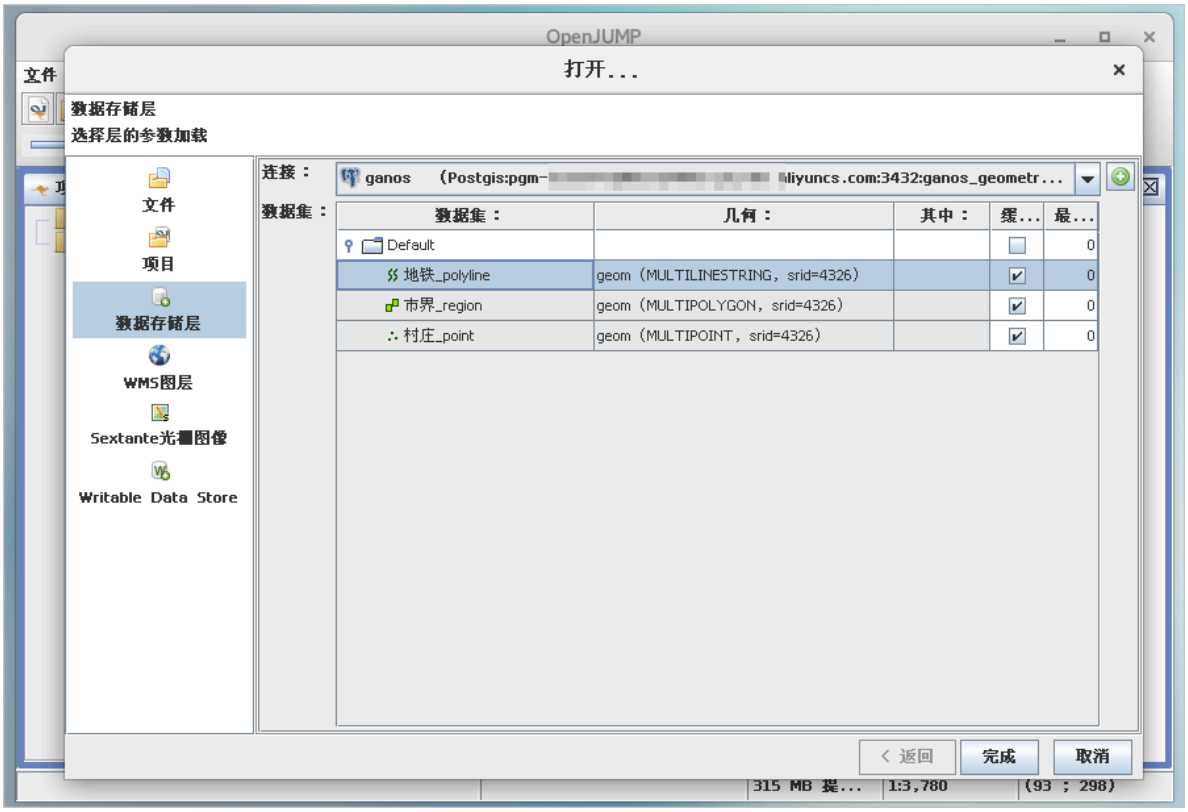
3. 在添加连接对话框中填入连接参数。



参数说明如下。

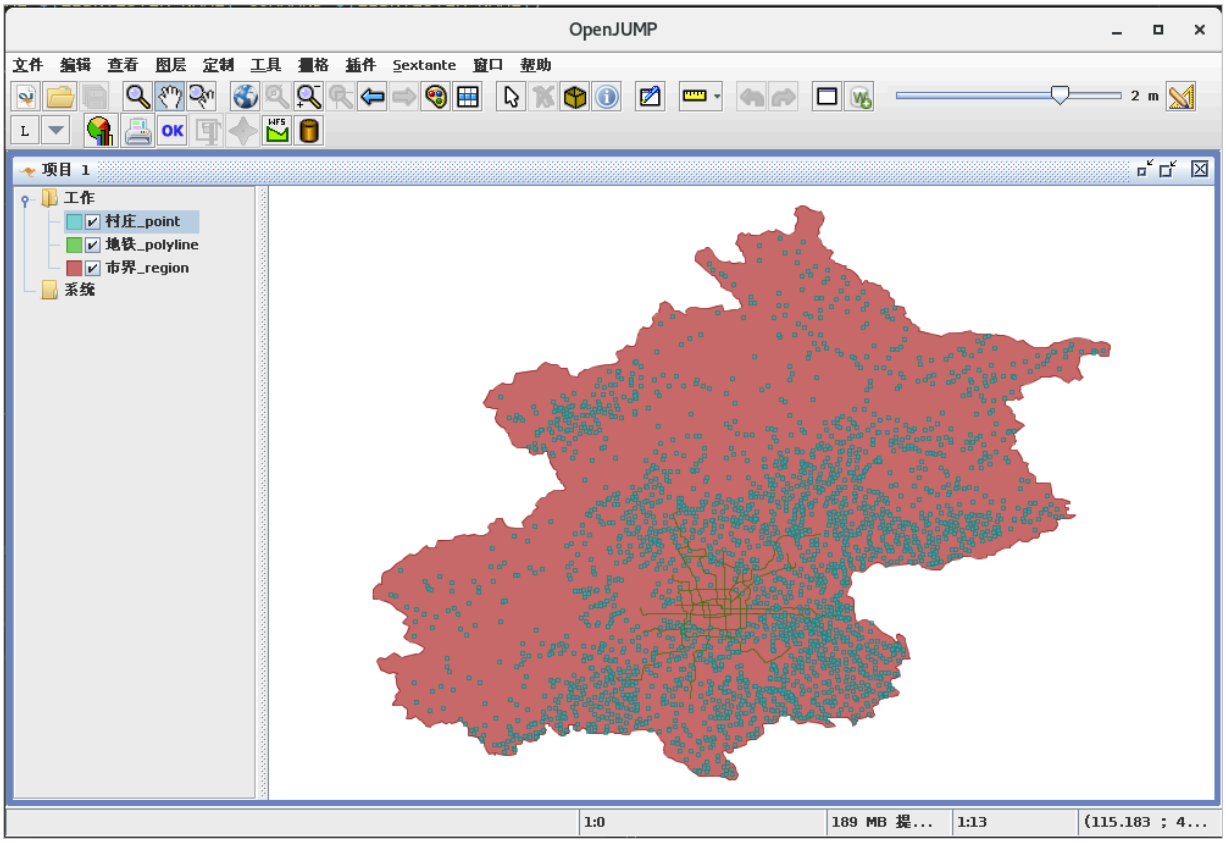
参数	说明
名称	自定义的连接名称。
驱动程序	指定为PostGIS。
Server	数据库实例的IP地址或RDS/POLARDB外网地址，可以从控制台中获得。
Port	数据库实例的端口地址，可以从控制台中获得。
Database	需要连接的数据库名。
User	数据库用户名。
Password	密码。

4. 单击确定后可以看到ganos库中所有的图层信息。



数据浏览与操作

连接后可以对数据进行浏览，可以进行放大，缩小，漫游等基本操作，也可以进行符号化渲染等。



也可以对几何对象进行属性查询。