ALIBABA CLOUD

阿里云

云原生数据仓库 AnalyticDB PostgreSQL 版

GANOS时空引擎扩展

文档版本：20220712

阿里云

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
危险	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。	危险 重置操作将丢失用户配置数据。
警告	该类警示信息可能会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，恢复业务时间约十分钟。
注意	用于警示信息、补充说明等，是用户必须了解的内容。	注意 权重设置为0，该服务器不会再接受新请求。
说明	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。	单击**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。	在**结果确认**页面，单击**确定**。
Courier字体	命令或代码。	执行 `cd /d C:/window` 命令，进入Windows系统文件夹。
斜体	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。	`switch {active|stand}`

目录

1.简介

AnalyticDB PostgreSQL版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AnalyticDB PostgreSQL版数据库中对空间或时空数据进行高效的存储、索引、查询和分析计算。本文介绍如何使用Ganos来对时空数据进行管理和分析。

空间/时空数据（Spatial/Spatio-temporal Data，以下统称时空数据）是带有时间或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。

> **说明** AnalyticDB PostgreSQL版Serverless模式暂不支持Ganos功能。

如果您需要获取人工帮助，可以拨打技术支持电话95187或者在控制台选择**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。

申明

本文中描述的部分产品特性或者服务可能不在您的购买或使用范围之内，请以实际商业合同和条款为准。本文档内容仅作为指导使用，文档中的所有内容不构成任何明示或暗示的担保。

定价

目前Ganos包含在AnalyticDB PostgreSQL版版本中，不单独计费。价格信息详情，请参见云原生数据仓库PostgreSQL版详细价格信息。

2.模型

2.1. 几何模型

简介

Ganos Geometry是ADB PG数据库的一个空间几何扩展。Ganos Geometry遵循OpenGIS规范，使ADB PG增加了存储管理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、空间几何索引、空间几何操作函数和空间几何操作符等丰富功能。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_spatialref;
Create extension ganos_geometry;
```

- 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,3857)) DIST
RIBUTED BY (ID);
```

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) )  DISTRIBUTED BY (ID);
SELECT AddGeometryColumn( 'roads', 'geom', 3857, '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)',3857),'北五环');
INSERT INTO roads (id, geom, road_name)
  VALUES (2,ST_GeomFromText('LINESTRING(189141 244158,189265 244817)',3857),'东五环');
INSERT INTO roads (id, geom, road_name)
  VALUES (3,ST_GeomFromText('LINESTRING(192783 228138,192612 229814)',3857),'南五环');
INSERT INTO roads (id, geom, road_name)
  VALUES (4,ST_GeomFromText('LINESTRING(189412 252431,189631 259122)',3857),'西五环');
INSERT INTO roads (id, geom, road_name)
  VALUES (5,ST_GeomFromText('LINESTRING(190131 224148,190871 228134)',3857),'东长安街');
INSERT INTO roads (id, geom, road_name)
  VALUES (6,ST_GeomFromText('LINESTRING(198231 263418,198213 268322)',3857),'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);
```

- 空间测量和空间分析

```
--Create Table bc_roads:
Column      | Type              | Description
------------+-------------------+-------------------
gid         | integer           | Unique ID
name        | character varying | Road Name
the_geom    | geometry          | Location Geometry (Linestring)
--Create table bc_municipality:
Column     | Type              | Description
-----------+-------------------+-------------------
gid        | integer           | Unique ID
code       | integer           | Unique ID
name       | character varying | City / Town Name
the_geom   | geometry          | Location Geometry (Polygon)
--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 GEOR
GE';
hectares
------------------
32657.9103824927
(1 row)
```

- 空间关系判断

```
--ST_Contains
```

```
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
...
--ST_Covers,a circle covering a circle
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 | bigcontainsexterior
--------------+----------------+-------------------+---------------------
 t            | f              | t                 | f
(1 row)
--ST_Disjoint
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)
--ST_Overlaps
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
--ST_Relate
SELECT ST_Relate(ST_GeometryFromText('POINT(1 2)'), ST_Buffer(ST_GeometryFromText('POINT(
1 2)'),2), '0FFFFF212');
st_relate
-----------
t
```

```
--ST_Touches
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)
--ST_Within
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)
```

- 几何对象存储

```
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)
```

- 删除扩展

```
Drop extension ganos_geometry;
DROP extension ganos_spatialref;
```

SQL参考

详细SQL请参见PostGIS官方手册。

2.2. 栅格模型

简介

栅格数据由按行和列（或格网）组织的像元（或像素）矩阵组成，其中的每个像元都包含一个信息值（例如温度）。

栅格数据可以是数字航空像片、卫星影像、数字图片或扫描的地图。

Ganos时空引擎通过在ADB PG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_spatialref;
Create extension ganos_geometry;
Create Extension Ganos_Raster;
```

- 创建栅格表

```
-- 指定raster 列作为表的 distributed column
Create Table raster_table(id integer, raster_obj raster)
DISTRIBUTED BY (raster_obj);
```

- 从OSS中导入栅格数据

```
Insert into raster_table
Values(1, ST_ImportFrom('chunk_table','OSS://<id>:<key>@oss-cn.aliyuncs.com/mybucket/data
/my_image.tif')),
(2, ST_CreateRast('OSS://<id>:<key>@oss-cn.aliyuncs.com/mybucket/data/my_image.tif'));
```

- 查询栅格对象信息

```
Select ST_Height(raster_obj),ST_Width(raster_obj) From raster_table Where id = 1;
 st_height
-----------
      1241
(1 rows)
```

- 创建金字塔

```
DO $$
declare
    rast raster;
begin
    select raster_obj into rast from raster_table where id = 1;
    rast = st_buildpyramid(rast);
    update raster_table set raster_obj = rast where id = 1;
end;
$$ LANGUAGE 'plpgsql';
DO $$
declare
    rast raster;
begin
    select raster_obj into rast from raster_table where id = 2;
    rast = st_buildpyramid(rast, -1, 'Near', 'chunk_table');
    update raster_table set raster_obj = rast where id = 2;
end;
$$ LANGUAGE 'plpgsql';
```

- 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(rast, '((128.0, 30.0),(128.5, 30.5))', 800, 600) from raster_t
able where id = 1;
---------------------
3
```

- 获取栅格指定范围像素矩阵

```
DO $$
declare
    rast raster;
begin
    select raster_obj into rast from raster_table where id = 1;
    rast = st_buildpyramid(rast);
    Select ST_Clip(rast, 0, '((128.980,30.0),(129.0,30.2))', 'World');
end;
$$ LANGUAGE 'plpgsql';
```

- 计算裁剪的像素坐标范围

```
Select ST_ClipDimension(raster_obj, 2, '((128.0, 30.0),(128.5, 30.5))')
from raster_table where id = 1;
-----------------------------------
'((600, 720),(200, 300))'
```

- 删除扩展

```
DROP Extension Ganos_Raster;
DROP extension ganos_geometry;
DROP extension ganos_spatialref;
```

SQL参考

详细SQL参考请参见Raster SQL参考。

2.3. 轨迹模型

简介

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进行深入研究。

Ganos Trajectory是对象关系型数据库ADB PG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。

快速入门

- 创建扩展

```
Create extension ganos_spatialref;
Create extension ganos_geometry;
Create Extension Ganos_trajectory;
```

- 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) DISTRIBUTED BY (id);
```

- 插入轨迹数据

```
insert into traj_table values
(1, ST_MakeTrajectory('STPOINT'::leaftype, st_geomfromtext('LINESTRING (114 35, 115 36, 1
16 37)', 4326), '[2010-01-01 14:30, 2010-01-01 15:30)'::tsrange, '{"leafcount": 3,"attrib
utes" : {"velocity" : {"type":"integer","length":4,"nullable":false,"value":[120, 130, 14
0]},"accuracy":{"type":"integer","length":4,"nullable":false,"value":[120, 130, 140]},"be
aring":{"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, 1
16 37)', 4326), '2010-01-01 14:30'::timestamp, '2010-01-01 15:30'::timestamp, '{"leafcoun
t": 3,"attributes" : {"velocity" : {"type":"integer","length":4,"nullable":false,"value":
[120, 130, 140]},"accuracy":{"type":"integer","length":4,"nullable":false,"value":[120, 1
30, 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, 1
16 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","lengt
h":4,"nullable":false,"value":[120, 130, 140]},"bearing":{"type":"float","length":4,"null
able":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, 1
16 37)', 4326), '[2010-01-01 14:30, 2010-01-01 15:30)'::tsrange, null));
```

- 创建轨迹索引

```
--创建轨迹索引，加速时空过滤效率
create index tr_index on traj_table using gist (traj);
--空间查询时，加速空间过滤
select id, traj_id from traj_test where st_3dintersects(traj, ST_GeomFromText('POLYGON((1
16.46747851805917 39.92317964155052,116.4986540687358 39.92317964155052,116.4986540687358
39.94452401711516,116.46747851805917 39.94452401711516,116.46747851805917 39.923179641550
52))'));
--时间查询时，加速时间过滤
select id, traj_id from traj_text where st_TContains(traj,'2008-02-02 13:30:44'::timestam
p,'2008-02-03 17:30:44'::timestamp);
--时空查询时，加速时空过滤
select id, traj_id from traj_test where st_3dintersects(traj, ST_GeomFromText('POLYGON((1
16.46747851805917 39.92317964155052,116.4986540687358 39.92317964155052,116.4986540687358
39.94452401711516,116.46747851805917 39.94452401711516,116.46747851805917 39.923179641550
52))'),'2008-02-02 13:30:44'::timestamp,'2008-02-03 17:30:44'::timestamp);
```

- 创建特定维度轨迹索引

```
--当我们只需要对轨迹进行特定维度分析时，可以只建立特定维度的索引。当我们不关心轨迹的z维时，可以使用tr
ajgist_op_2dt建立二维+时间索引
create index tr_timespan_time_index on traj_table using gist (traj trajgist_op_2dt);
--建立特定维度索引后对2维+时间查询效果会更快
select id, traj_id from traj_test where st_2dintersects(traj, ST_GeomFromText('POLYGON((1
16.46747851805917 39.92317964155052,116.4986540687358 39.92317964155052,116.4986540687358
39.94452401711516,116.46747851805917 39.94452401711516,116.46747851805917 39.923179641550
52))'),'2008-02-02 13:30:44'::timestamp,'2008-02-03 17:30:44'::timestamp);
--可以同时有多个索引存在，建立任意一个即可支持所有查询，当有多个时将自动选择最优的索引
create index tr_timespan_time_index on trajtab using gist (traj trajgist_op_2d);
select id, traj_id from traj_test where st_2dintersects(traj, ST_GeomFromText('POLYGON((1
16.46747851805917 39.92317964155052,116.4986540687358 39.92317964155052,116.4986540687358
39.94452401711516,116.46747851805917 39.94452401711516,116.46747851805917 39.923179641550
52))'));
```

- 查询轨迹

```
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

```
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_Raster;
DROP extension ganos_geometry;
Drop Extension Ganos_trajectory;
```

SQL参考

详细SQL参考请参见Trajectory SQL参考。

3.Raster SQL参考

3.1. 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像元，并在各像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。栅格的左上角点作为起始原点。
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。

3.2. Raster创建

3.2.1. 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);
raster ST_CreateRast(cstring url, cstring storageOption);
```

参数

参数名称	描述
url	OSS影像文件的路径。更多信息，请参见对象存储服务路径。
storageOption	基于JSON格式的字符串，描述raster对象金字塔的分块存储信息。

storageOption支持的参数如下：

参数名称	描述	类型	格式	默认值	说明
chunkdim	分块的维度信息	string	(w, h, b)	从原始影像中读取分块大小	无
interleaving	交错方式	string	无	bsq	必须是以下一种： • bip：像素交错 • bil：行交错 • bsq：波段交错 • auto：根据原始影像自动选择

> **说明** 在通常情况下无需修改分块和交错信息，仅在特殊场景下需要修改，例如：
> - 需要多波段RGB组合浏览，但是默认值为bsq（波段交错），需要修改为bip（像素交错）。
> - 某些影像的分块大小为1行*n列，但是访问请求为256行*256列的规则块，需要修改为按照访问请求的大小。

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ADB PG与OSS所在地域相同，详情请参见OSS Endpoint。

示例

```
-- 基于OSS存储，指定accessID,accessKEY,endpoint
Select ST_CreateRast('OSS://<ak>:<ak_secret>@oss-cn-beijing-internal.aliyuncs.com/mybucket/data/image.tif');
-- 基于MinIO,指定host和port
Select ST_CreateRast('OSS://<ak>:<ak_secret>@10.0.0.1:443/mybucket/data/image.tif');
-- 指定分块与交错方式
Select ST_CreateRast('OSS://<ak>:<ak_secret>@oss-cn-beijing-internal.aliyuncs.com/mybucket/data/image.tif', '{"chunkdim":"(256,256,3)","interleaving":"auto"}');
-- 指定具有Subset的NetCDF对应的影像
Select ST_CreateRast('OSS://<ak>:<ak_secret>@oss-cn-beijing-internal.aliyuncs.com/mybucket/data/image.nc:hcc');
```

3.3. 导入导出

3.3.1. ST_ImportFrom

从一个OSS文件导入到数据库。

语法

```
raster ST_ImportFrom(cstring chunkTableName,
                     cstring url,
                     cstring storageOption default '{}',
                     cstring importOption default '{}');
```

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。详情请请参见ST_CreateRast中构建路径的描述。

描述

支持导入的数据类型如下：

名称	全称
BMP	Microsoft Windows Device Independent Bitmap(.bmp)
EHdr	ESRI .hdr Labelled
ENVI	ENVI .hdr Labelled Raster
GTiff	TIFF/BigTIFF/GeoTIFF(.tif)
HFA	Erdas Imagine .img
RST	Idrisi Raster Format
INGR	Intergraph Raster Format
NetCDF	Network Common Data Form
AAIGrid	Arc/Info ASCII Grid
AIG	Arc/Info Binary Grid
GIF	Graphics Interchange Format
PNG	Portable Network Graphics (.png)

名称	全称
JPEG	JPEG JFIF File Format

示例

```
Select ST_ImportFrom('chunk_table','OSS://<ak>:<ak_secret>@oss-cn-beijing-internal.aliyuncs
.com/mybucket/data/image.tif');
-- 指定具有Subset的NetCDF对应的影像
Select ST_ImportFrom('chunk_table','OSS://<ak>:<ak_secret>@oss-cn-beijing-internal.aliyuncs
.com/mybucket/data/image.nc:hcc');
```

3.3.2. 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。详情请参见ST_CreateRast 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)
EHdr	ESRI .hdr Labelled
ENVI	ENVI .hdr Labelled Raster
GTiff	TIFF/BigTIFF/GeoTIFF(.tif)

名称	全称
HFA	Erdas Imagine .img
RST	Idrisi Raster Format
INGR	Intergraph Raster Format

示例

```
DO $$
declare
    rast raster;
begin
    select raster_obj into rast from raster_table where id = 1;
    Select ST_ExportTo(rast, 'GTiff', 'OSS://ABCDEFG:1234567890@oss-cn-beijing-internal.aliyuncs.com/mybucket/data/4.tif');
end;
$$ LANGUAGE 'plpgsql';
```

3.4. 金字塔操作

3.4.1. ST_BuildPyramid

创建影像金字塔。

语法

```
raster ST_BuildPyramid(raster source,
                       integer pyramidLevel default -1,
                       ResampleAlgorithm algorithm default 'Near',
                       cstring chunkTableName default '',
                       cstring storageOption default '{}',
                       cstring buildOption default '{}');
```

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。只对基于对象存储（OSS）的栅格对象有效。
pyramidLevel	金字塔创建的层级， -1表示创建到最高层级。

参数名称	描述
algorithm	创建金字塔的重采样算法，取值如下： • Near：最邻近 • Average：平均值 • Bilinear：二次线性 • Cubic：三次卷积
storageOption	JSON字符串，存储选项。描述raster对象金字塔的分块存储信息。该选项只针对基于对象存储OSS的栅格对象有效。
buildOption	JSON字符串，构建选项。当前支持参数parallel，可以设置操作并行度，数据类型为Integer，取值范围为1~64。不指定parallel时，使用GUC参数ganos.parallel.degree的值。 **说明** 如果启用并行创建金字塔，则不支持事务。如果创建失败或需要对事务回滚，使用ST_deletePyramid删除已经创建的金字塔。

storageOption参数说明如下。

参数名称	类型	说明
chunkdim	string	分块的维度信息，格式为 `(w, h, b)` ，默认从原始影像中读取分块大小。
interleaving	string	交错方式，取值如下： • bip：像素交错 • bil：行交错 • bsq（默认值）：波段交错 • auto：根据原始影像自动选择
compression	string	压缩算法类型，取值如下： • none • jpeg • zlib • png • lzo • lz4（默认值） • zstd • snappy • jp2k
quality	integer	压缩质量。只针对jpeg和jp2k压缩算法生效，默认值为75。

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

```
DO $$
declare
    rast raster;
begin
    select raster_obj into rast from raster_table where id = 1;
    rast = st_buildpyramid(rast);
    update raster_table set raster_obj = rast where id = 1;
end;
$$ LANGUAGE 'plpgsql';
DO $$
declare
    rast raster;
begin
    select raster_obj into rast from raster_table where id = 1;
    rast = st_buildpyramid(rast, 'chunk_table');
    update raster_table set raster_obj = rast where id = 1;
end;
$$ LANGUAGE 'plpgsql';
```

3.4.2. 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

```
DO $$
declare
    rast raster;
begin
    select raster_obj into rast from raster_table where id = 1;
    rast = ST_deletePyramid(rast);
    update raster_table set raster_obj = rast where id = 1;
end;
$$ LANGUAGE 'plpgsql';
```

3.4.3. 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 rast
er_table where id = 10;
---------------------
3
```

3.5. 坐标系统转换

3.5.1. 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 col
umn);
geometry ST_Rast2WorldCoord(raster raster_obj, integer pyramidLevel,  geometry geom);
```

参数

参数名称	描述
raster_obj	目标raster对象

参数名称	描述
pyramidLevel	金字塔层级
row	行号
column	列号
geom	需要转换的几何对象，横坐标x值表示象元的列号，纵坐标y值表示象元的行号

描述

raster对象必须要有完整的空间参考信息。

示例

```
SELECT ST_rast2WorldCoord(raster_obj, 0, 3, 4) FROM raster_table;
 st_rast2worldcoord
--------------------
 (440960,3751140)
 SELECT ST_AsText(ST_rast2WorldCoord(raster_obj, 0, 'POINT(4 3)'::geometry)) FROM raster_ta
ble;
       st_astext
------------------------
 POINT(440960 3751140)
```

3.5.2. 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);
geometry ST_World2RastCoord(raster raster_obj,  integer pyramidLevel, geometry geom);
```

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。
geom	需要转换的几何对象。

描述

raster对象必须要有完整的空间参考信息。

返回的几何对象，横坐标x值表示象元的列号，纵坐标y值表示象元的行号。

示例

```
select st_world2rastcoord(rast, 0, '(117.3378,26.9020)'::point) from tb_dem where id = 2;
 st_world2rastcoord
--------------------
 (53205,32518)
SELECT ST_AsText(ST_world2RastCoord(rast, 0, ST_Rast2WorldCoord(rast, 0, 'POINT(511 0)'::ge
ometry)))
FROM tb_world2rast;
  st_astext
--------------
 POINT(511 0)
```

3.6. 像素值操作

3.6.1. 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。
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 w
here id = 10;
-----------------------------------
'((200, 300),(600, 720))'
```

3.6.2. 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 '',
                    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	指定输出像元子集的空间参考值。
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。
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	表示裁剪选项。
storageOption	表示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。

参数名称	类型	默认值	描述
rast_coord	bool	false	传入的geometry是否使用的是象元坐标。如果是象元坐标，横坐标x表示象元的列号（起始为0），纵坐标y表示象元的行号（起始为0）。

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100 MB，代表最多只能裁剪出100 MB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

```
DO $$
declare
    rast raster;
begin
    select raster_obj into rast from raster_table where id = 1;
    Select ST_Clip(rast, 0, '((128.980,30.0),(129.0,30.2))', 'World');
end;
$$ LANGUAGE 'plpgsql';
```

3.6.3. 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 '',
                     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。
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。
rast_coord	bool	false	传入的Geometry是否使用的是象元坐标。如果是象元坐标，横坐标x表示象元的列号，纵坐标y表示象元的行号。

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。

参数名称	类型	默认值	描述
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：波段按像元交叉 • bil：波段按行交叉 • bsq：波段顺序
endian	string	和原始Raster一致	字节序。取值： • NDR：小字节序（Little endian） • XDR：大字节序（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100 MB，代表最多只能裁剪出100 MB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

```
DO $$
declare
    rast raster;
    new_rast raster;
begin
    -- 永久表
    CREATE TEMP TABLE rast_clip_result(id integer, rast raster);
    select raster_obj into rast from raster_table where id = 1;
    new_rast = ST_ClipToRast(rast, ST_geomfromtext('Polygon((0 0, 45 45, 90 45, 45 0, 0 0))
', 4326), 0);
    Insert into rast_clip_result values(1, new_rast);
end;
$$ LANGUAGE 'plpgsql';
```

3.6.4. ST_AddZ

根据栅格的波段值来对geometry的z值进行设置。

语法

```
geometry ST_AddZ(raster source,
                 geometry geom,
                 integer pyramid,
                 integer band);
```

参数

参数名称	描述
source	需要计算的raster对象。
geom	需要查询的几何对象。
pyramid	栅格的金字塔层级值，从0开始，金字塔层级为N，金字塔层级值为0~N-1中的整数，默认值为0。
band	栅格的波段值，从0开始，波段数量为N时，波段值为0~N-1中的整数，默认值为0。

描述

根据栅格的波段值设置geometry的z值。如果栅格设置了几何参考，几何对象按照地理坐标进行查询，否则按照像元坐标进行查询。

> **说明** 几何对象上的点必须完全在栅格对象内。

示例

```
DO $$
declare
    rast raster;
begin
    select raster_obj into rast from raster_table where id = 1;
    SELECT ST_AddZ(rast, ST_GeomFromText('POINT(120.5 30.6)', 4326), 0, 0);
end;
$$ LANGUAGE 'plpgsql';
```

3.6.5. 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。

涉及的数据库参数如下。

参数	类型	说明
ganos.raster.mosaic_must_same_nodata	boolean	指定镶嵌时数据源的nodata值是否必须一致。取值：true|false。镶嵌时并不会对nodata值进行转换，如果选择可以不一致（false），可能会导致镶嵌后的像素语义不一致。示例： `Set ganos.raster.mosaic_must_same_nodata = false;`

示例

```
INSERT INTO raster_table 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_tabl
e WHERE id < 10), 'chunk_table_mosaic') WHERE id = 11;
```

3.7. DEM操作

3.7.1. 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 typ
e, 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

------------------------------------------------------------------------------------------------
-----------------------------
 {0.210279822382945,0.369954478614613,0.220241089748741,0.415504885724335,0.523380429142649
,0.19079849696355,0.32493941.
.6771785,0.592806538904308,0.559435506670953,0.487598684366856,0.17107200555711,0.084021792
2621739,0.381860307736563,0..
.580949493078414,0.638382145824945,0.43822706997925,0.317039337499457,0.284095352866283,0.2
84298114561498,0.49448931974.
.7884,0.817870125632039,0.645927342349833,0.241209216517688,0.211549813235801,0.33904046395
4188,0.636582806346833,0.934.
.672430381381,0.7814534477193,0.0832677675316725,0.0544326656785603,0.529537557031012,0.836
305912538514,0.9446346707062.
.92,0.747858756227042,0.0284106287186713,0.0616861154423774}
(1 row)
```

3.8. 属性查询与更新

3.8.1. 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
```

3.8.2. 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)
```

3.8.3. ST_MetaData

获得raster对象的元数据，返回JSON格式。

语法

```
text ST_MetaData(raster raster_obj);
text ST_MetaData(raster raster_obj,
                 text key);
text ST_MetaData(raster raster_obj,
                 integer band,
                 text key);
```

参数

参数名称	描述
raster_obj	raster对象。
band	波段序号，从0开始。
key	需要查询的元数据项名称，如果传入的是 `all` ，则返回一个包含所有元数据项的JSON。

示例

```
select ST_MetaData(raster_obj) from raster_table;
-- meta data with a name
SELECT ST_MetaData(raster_obj, 'swh#scale_factor')
FROM raster_table;
      st_metadata
-----------------------
 0.0001488117874873806
-- all meta data
SELECT ST_MetaData(raster_obj, 'all')
FROM raster_table;
      st_metadata
-----------------------
{"AREA_OR_POINT":"Area"}
-- meta data with a name of a band
SELECT ST_MetaData(raster_obj, 0, 'NETCDF_DIM_time')
FROM raster_table;
 st_metadata
-------------
 1043112
 -- all meta data of a band
SELECT ST_MetaData(raster_obj, 'all')
FROM raster_table;
                                                                st_metadata

-----------------------------------------------------------------------------------------------------
-------------------------------------------
{"add_offset":"4.907141431495487","long_name":"Significant height of combined wind waves an
d swell","missing_value":"-32767","NETCDF_DI.
M_time":"1043112","NETCDF_VARNAME":"swh","scale_factor":"0.0001488117874873806","units":"m"
,"_FillValue":"-32767"}
```

3.8.4. ST_Width

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

语法

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

参数

参数名称	描述
raster_obj	raster对象。
pyramid	金字塔层级，从0开始。

示例

```
select ST_Width(raster_obj) from raster_table;
————————————————————
10060
```

3.8.5. ST_Height

获得raster对象的高度。

语法

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

参数

参数名称	描述
raster_obj	raster对象。
pyramid	金字塔层级，从0开始。

示例

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

3.8.6. 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
```

3.8.7. 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_no
data_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_nodata_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)
```

3.8.8. 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
```

3.8.9. 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
```

3.8.10. 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
```

3.8.11. 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(raster raster_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
```

3.8.12. 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
```

3.8.13. ST_Extent

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

语法

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

参数

参数名称	描述
raster_obj	raster对象。
csOption	坐标空间选项枚举值之一。
pyramid	金字塔层级，从0开始。

描述

csOption为坐标空间选项，取值如下：

- 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，即如果已进行地理参考，则返回世界坐标，如果未进行地理参考，则返回像元坐标。

示例

```
select ST_Extent(raster_obj, 'Raster'::CoorSpatialOption) from raster_table;
_____________________________________
((255, 255), (0, 0))
```

3.8.14. ST_ConvexHull

根据栅格的地理参考信息获得栅格对象的凸包。

语法

```
geometry ST_ConvexHull(raster source);
geometry ST_ConvexHull(raster source,
                       integer pyramid);
```

参数

参数名称	描述
source	需要计算的raster对象。
pyramid	金字塔层级，从0开始，默认值为0。

描述

如下图所示，红色外框包含的部分为栅格对象的凸包。

示例

```
SELECT st_astext(st_convexhull(rast_object))
FROM raster_table;
----------------------------------------------------
 POLYGON((-180 90,180 90,180 -90,-180 -90,-180 90))
-- 指定金字塔层级
SELECT st_astext(st_convexhull(rast_object,  1))
FROM raster_table;
----------------------------------------------------
 POLYGON((-180 90,180 90,180 -90,-180 -90,-180 90))
```

3.8.15. ST_Envelope

根据栅格的地理参考信息获得栅格对象的外包矩形。

语法

```
geometry ST_Envelope(raster source);
geometry ST_Envelope(raster source,
                     integer pyramid);
```

参数

参数名称	描述
source	需要计算的raster对象。
pyramid	金字塔层级，从0开始，默认值为0。

描述

如下图所示，红色外框包含的部分为栅格对象的外包矩形。

示例

```
SELECT st_astext(st_envelope(rast_object))
FROM raster_table;
----------------------------------------------------
 POLYGON((-180 90,180 90,180 -90,-180 -90,-180 90))
-- 指定金字塔层级
SELECT st_astext(st_envelope(rast_object,  1))
FROM raster_table;
----------------------------------------------------
 POLYGON((-180 90,180 90,180 -90,-180 -90,-180 90))
```

3.8.16. 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
```

3.8.17. 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)
```

3.8.18. ST_ScaleX

获得raster对象在空间参考系下X方向的像素宽度。

前提条件

raster对象必须已经进行空间参考。

语法

```
float8 ST_ScaleX(raster raster_obj)
```

参数

参数名称	描述
raster_obj	raster对象。

示例

```
select st_scalex(rast), st_scaley(rast)
from raster_table;
 st_scalex | st_scaley
-----------+-----------
       1.3 |       0.3
```

3.8.19. ST_ScaleY

获得raster对象在空间参考系下Y方向的像素宽度。

前提条件

raster对象必须已经进行空间参考。

语法

```
float8 ST_ScaleY(raster raster_obj)
```

参数

参数名称	描述
raster_obj	raster对象。

示例

```
select st_scalex(rast), st_scaley(rast)
from raster_table;
 st_scalex | st_scaley
-----------+-----------
       1.3 |       0.3
```

3.8.20. ST_SetScale

设置raster对象在空间参考系下X、Y方向的像素宽度。

语法

```
raster ST_SetScale(raster raster_obj,  float8 scaleX, float8 scaleY)
raster ST_SetScale(raster raster_obj,  float8 scaleXY)
```

参数

参数名称	描述
raster_obj	raster对象。
scaleX	空间参考系下X方向像素宽。
scaleY	空间参考系下Y方向像素宽。
scaleXY	空间参考系下X、Y方向像素宽（X、Y为相同值）。

示例

```
UPDATE raster_table
SET rast = ST_SetScale(rast, 1.3, 0.6)
WHERE id = 2;
select st_scalex(rast), st_scaley(rast)
from raster_table
WHERE id = 2;
 st_scalex | st_scaley
-----------+-----------
       1.3 |       0.6
```

3.8.21. ST_SkewX

获得raster对象在X方向的旋转。

前提条件

raster对象必须已经进行空间参考。

语法

```
float8 ST_SkewX(raster raster_obj)
```

参数

参数名称	描述
raster_obj	raster对象。

示例

```
select st_skewx(rast), st_skewy(rast)
from raster_table;
 st_skewx | st_skewy
----------+-----------
      0.3 |       0.2
```

3.8.22. ST_SkewY

获得raster对象在Y方向的旋转。

前提条件

raster对象必须已经进行空间参考。

语法

```
float8 ST_SkewY(raster raster_obj)
```

参数

参数名称	描述
raster_obj	raster对象。

示例

```
select st_skewx(rast), st_skewy(rast)
from raster_table;
 st_skewx | st_skewy
----------+----------
      0.3 |      0.2
```

3.8.23. ST_SetSkew

设置raster对象X、Y方向的旋转。

语法

```
raster ST_SetSkew(raster raster_obj,  float8 skewX, float8 skewY)
raster ST_SetSkew(raster raster_obj,  float8 skewXY)
```

参数

参数名称	描述
raster_obj	raster对象。
skewX	X方向的旋转。
skewY	Y方向的旋转。
skewXY	X、Y方向的旋转（X、Y为相同值）。

示例

```
UPDATE raster_table
SET rast = ST_SetSkew(rast, 0.3, 0.6)
WHERE id = 2;
select st_skewx(rast), st_skewy(rast)
from raster_table
WHERE id = 2;
 st_skewx | st_skewy
----------+----------
      0.3 |      0.6
```

3.8.24. ST_UpperLeftX

获得raster对象在空间参考系下左上角点X值。

前提条件

raster对象必须已经进行空间参考。

语法

```
float8 ST_UpperLeftX(raster raster_obj)
```

参数

参数名称	描述
raster_obj	raster对象。

示例

```
select st_upperleftx(rast), st_upperlefty(rast)
from raster_table;
 st_upperleftx | st_upperlefty
---------------+---------------
        440720 |       3751320
```

3.8.25. ST_UpperLeftY

获得raster对象在空间参考系下左上角点Y值。

前提条件

raster对象必须已经进行空间参考。

语法

```
float8 ST_UpperLeftY(raster raster_obj)
```

参数

参数名称	描述
raster_obj	raster对象。

示例

```
select st_upperleftx(rast), st_upperlefty(rast)
from raster_table;
 st_upperleftx | st_upperlefty
---------------+---------------
        440720 |       3751320
```

3.8.26. ST_SetUpperLeft

设置raster对象在空间参考系下左上角点X、Y值。

语法

```
raster ST_SetUpperLeft(raster raster_obj,  float8 upperleftX, float8 upperleftY)
raster ST_SetUpperLeft(raster raster_obj,  float8 upperleftXY)
```

参数

参数名称	描述
raster_obj	raster对象。
upperleftX	空间参考系下左上角点X。
upperleftY	空间参考系下左上角点Y。
upperleftXY	空间参考系下左上角点X、Y值（X、Y为相同值）。

示例

```
UPDATE raster_table
SET rast = ST_SetUpperleft(rast, 120, 30)
WHERE id = 2;
select st_upperleftx(rast), st_upperlefty(rast)
from raster_table
WHERE id = 2;
 st_upperleftx | st_upperlefty
---------------+---------------
          120  |            30
```

3.8.27. ST_PixelWidth

获得raster对象在空间参考系下像素的宽度。

语法

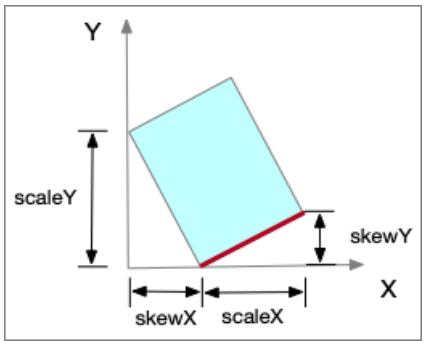
```
float8 ST_PixelWidth(raster raster_obj)
```

参数

参数名称	描述
raster_obj	raster对象。

描述

PixelWidth如下图红色部分所示。如果raster的旋转为0，即等同于ScaleX。



示例

```
select st_pixelwidth(rast), st_pixelheight(rast)
from raster_table;
 st_pixelwidth | st_pixelheight
---------------+----------------
            60 |             60
```

3.8.28. ST_PixelHeight

获得raster对象在空间参考系下像素的高度。

前提条件

raster对象必须已经进行空间参考。

语法

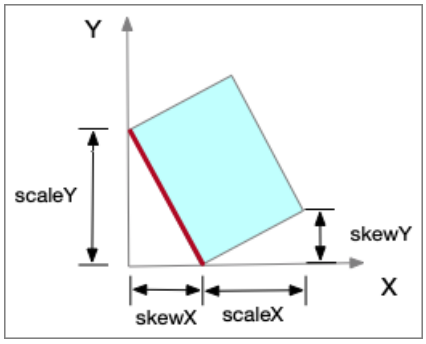
```
 float8 ST_PixelHeight(raster raster_obj)
```

参数

参数名称	描述
raster_obj	raster对象。

描述

PixelHeight如下图红色部分所示。如果raster的旋转为0，即等同于ScaleY。



示例

```
select st_pixelwidth(rast), st_pixelheight(rast)
from raster_table;
 st_pixelwidth | st_pixelheight
---------------+----------------
            60 |             60
```

3.8.29. 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
```

3.8.30. 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
```

3.8.31. 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)
```

3.8.32. 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, doubl
e 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)
```

3.8.33. 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_sn)
```

参数

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

示例

```
select ST_NoData(raster_obj, 0) from raster_table where id=1;
_____________________________________
0.000
```

3.8.34. 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)
```

3.8.35. 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}
    ]
}'
```

3.8.36. 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，参见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)
```

3.8.37. 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}'
```

3.8.38. 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, dou
ble 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 {"approx":true}。

示例

```
update rast set rast=ST_SetStatistics(rast,0,0.0 , 255.0, 125.0, 23.6, '{"approx":false}');
_____________________________________
(1 row)
```

3.8.39. 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)
raster ST_SummaryStats(raster raster_obj, cstring statsOption)
raster ST_SummaryStats(raster raster_obj,
                       cstring bands,
                       cstring statsOption)
```

参数

参数名称	描述
raster_obj	raster对象。
bands	指定的波段序号。从0开始，格式为 `'0'` 、 `'1-3'` 或 `'1,2,3'` 形式。
statsOptions	统计值选项JSON字符串。

statsOptions用于指定统计参数，参数如下：

参数名称	描述	类型	格式	默认值	说明
approx	是否使用采样方式计算统计值。	boolean	无	true	• true：采样计算统计值，结果可能会不精确。 • false：计算所有统计值。

示例

```
DO $$
declare
    rast raster;
begin
    select raster_obj into rast from raster_table where id = 1;
    rast = ST_SummaryStats(rast);
    update raster_table set raster_obj = rast where id = 1;
end;
$$ LANGUAGE 'plpgsql';
```

3.8.40. 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。
RGBIndex	RGB颜色模型颜色表索引。
RGBAIndex	RGBA颜色模型颜色表索引。
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。

值	说明
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
```

3.8.41. 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。

值	说明
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。
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)
```

3.8.42. 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,
1504,1590,1786,1870,2044,2099,2277,2373,2451,2585,2646,2882,2878,3091,3396,3620,3911,4124,4
304,4700,4893,5314,5446,5657,5765,5649,5749,5753,5601,5335,5161,4943,4592,4445,4207,4083,40
90,4270,4465,4514,4844,5204,5331,5597,5777,5838,6004,6316,6095,5762,5567,5465,4923,4677,422
0,3843,3401,3041,2571,2345,1972,1725,1376,1140,1008,841,716,548,442,373,308,212,133,78,68,3
1,24,12,2,2,1],
 "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,
116.0,117.0,118.0,119.0,120.0,121.0,122.0,123.0,124.0,125.0,126.0,127.0,128.0,129.0,130.0,1
31.0,132.0,133.0,134.0,135.0,136.0,137.0,138.0,139.0,140.0,141.0,142.0,143.0,144.0,145.0,14
6.0,147.0,148.0,149.0,150.0,151.0,152.0,153.0,154.0,155.0,156.0,157.0,158.0,159.0,160.0,161
.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]
  }
 }
}
```

3.8.43. 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	-
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/m axValue	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)
```

3.8.44. 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

```
DO $$
declare
    rast raster;
begin
    select raster_obj into rast from raster_table where id = 1;
    rast = st_buildhistogram(rast);
    update raster_table set raster_obj = rast where id = 1;
end;
$$ LANGUAGE 'plpgsql';
```

3.8.45. ST_StatsQuantile

计算raster对象的中位数。

语法

```
raster ST_StatsQuantile(raster raster_obj)
```

参数

参数名称	描述
raster_obj	raster对象。

描述

按照波段计算中位数，计算结果会记录到raster对象的元数据中。

示例

```
DO $$
declare
    rast raster;
begin
    select raster_obj into rast from raster_table where id = 1;
    rast = ST_StatsQuantile(rast);
    update raster_table set raster_obj = rast where id = 1;
end;
$$ LANGUAGE 'plpgsql';
```

3.8.46. ST_Quantile

查询raster对象分位数的像素值。

前提条件

通过ST_StatsQuantile预先计算分位数。

语法

```
setof record ST_Quantile(raster raster_obj,
                  float8[] quantiles default NULL,
                  cstring bands default '',
                  boolean exclude_nodata_value default true,
                  out integer band,
                  out float8 quantile,
                  out float8 value)
```

参数

参数名称	描述
raster_obj	raster对象。
quantiles	需要计算的分位数，取值为0.25、0.5和0.75中的一个或多个。
bands	需要计算的波段，格式为 `'0-2'` 或者 `'1,2,3'` ，从0开始。 默认为 `''` ，表示裁剪所有的波段。
exclude_nodata_value	是否需要计算nodata。
band	返回波段号。
quantile	返回分位数。
value	返回像素值。

示例

```
-- 计算所有波段 0.25 分位数的像素值。
SELECT  (ST_Quantile(rast, ARRAY[0.25], '0-2', true)).* FROM rat_quantile WHERE id = 1;
 band | quantile | value
------+----------+-------
    0 |     0.25 |    11
    1 |     0.25 |    10
    2 |     0.25 |    50
(3 rows)
-- 计算0波段所有分位数的像素值。
SELECT  (ST_Quantile(rast, NULL, '0', true)).* FROM rat_quantile WHERE id = 1;
 band | quantile | value
------+----------+-------
    0 |     0.25 |    11
    0 |      0.5 |    11
    0 |     0.75 |    65
(3 rows)
```

3.8.47. ST_MD5Sum

返回一个raster对象的MD5字符串。

语法

```
text ST_MD5Sum(raster raster_obj)
```

参数

参数名称	描述
raster_obj	raster对象。

描述

先试图从元数据中获得MD5字符串，如果元数据中不存在，数据内部raster对象返回NULL，然后OSS存储的raster对象再根据OSS路径进行计算。

GUC ganos.raster.calculate_md5：指定是否在raster对象入库时计算md5sum并保存到元数据中，默认值是false。有关变量信息请参见ganos.raster.calculate_md5。

GUC ganos.raster.md5sum_chunk_size：指定计算md5时每次读入的缓存的大小，默认值是10MB。有关变量信息请参见ganos.raster.md5sum_chunk_size。

示例

```
SELECT ST_MD5SUM(rast)
FROM raster_table
WHERE id = 1;
              st_md5sum
----------------------------------
 21f41fd983d3139c75b04bff2b7bf5c9
```

3.8.48. ST_SetMD5Sum

设置raster对象的MD5字符串。

语法

```
text ST_SetMD5Sum(raster raster_obj, text md5sum)
```

参数

参数名称	描述
raster_obj	raster对象。
md5sum	MD5字符串。

描述

将MD5字符串记录到raster对象中。MD5字符串长度必须为32位，并且只能由数字和小写字母组成。

GUC ganos.raster.calculate_md5：指定是否在raster对象入库时计算md5sum并保存到元数据中。有关变量信息请参见ganos.raster.calculate_md5。

示例

```
SELECT ST_MD5SUM(ST_SetMD5Sum(rast,'21f41fd983d3139c75b04bff2b7bf5c9'))
FROM raster_table
WHERE id = 1;
              st_md5sum
----------------------------------
 21f41fd983d3139c75b04bff2b7bf5c9
```

3.8.49. ST_XMin

获得raster对象在X方向的最小值。

语法

```
float8 ST_XMin(raster raster_obj)
float8 ST_XMin(raster raster_obj, integer pyramid)
```

参数

参数名称	描述
raster_obj	raster对象。
pyramid	金字塔层级，从0开始。

描述

如果raster对象已经进行地理参考，返回的是X坐标的最小值，否则返回0。

示例

```
select ST_XMin(rast)
from raster_table;
 st_xmin
----------
      120
```

3.8.50. ST_YMin

获得raster对象在Y方向的最小值。

语法

```
float8 ST_YMin(raster raster_obj)
float8 ST_YMin(raster raster_obj, integer pyramid)
```

参数

参数名称	描述
raster_obj	raster对象。
pyramid	金字塔层级，从0开始。

描述

如果raster对象已经进行地理参考，返回的是Y坐标的最小值，否则返回0。

示例

```
select ST_YMin(rast)
from raster_table;
 st_ymin
----------
      30
```

3.8.51. ST_XMax

获得raster对象在X方向的最大值。

语法

```
float8 ST_XMax(raster raster_obj)
float8 ST_XMax(raster raster_obj, integer pyramid)
```

参数

参数名称	描述
raster_obj	raster对象。
pyramid	金字塔层级，从0开始。

描述

如果raster对象已经进行地理参考，返回的是X坐标的最大值，否则返回raster对象宽度-1。

示例

```
select ST_XMax(rast)
from raster_table;
 st_xmax
----------
      121
```

3.8.52. ST_YMax

获得raster对象在Y方向的最大值。

语法

```
float8 ST_YMax(raster raster_obj)
float8 ST_YMax(raster raster_obj, integer pyramid)
```

参数

参数名称	描述
raster_obj	raster对象。
pyramid	金字塔层级，从0开始。

描述

如果raster对象已经进行地理参考，返回的是Y坐标的最大值，否则返回raster对象高度-1。

示例

```
select st_ymax(rast)
from raster_table;
 st_ymax
----------
      31
```

3.8.53. ST_ChunkHeight

获得raster对象分块的高度。

语法

```
integer ST_ChunkHeight(raster raster_obj)
```

参数

参数名称	描述
raster_obj	raster对象。

描述

获得raster对象分块的高度，单位：像素。

示例

```
select ST_ChunkHeight(rast)
from raster_table;
 st_chunkheight
----------
      256
```

3.8.54. ST_ChunkWidth

获得raster对象分块的宽度。

语法

```
integer ST_ChunkWidth(raster raster_obj)
```

参数

参数名称	描述
raster_obj	raster对象。

描述

获得raster对象分块的宽度，单位：像素。

示例

```
select ST_ChunkWidth(rast)
from raster_table;
 st_chunkwidth
----------
      256
```

3.8.55. ST_ChunkBands

获得raster对象分块波段维数量。

语法

```
integer ST_ChunkBands(raster raster_obj)
```

参数

参数名称	描述
raster_obj	raster对象。

示例

```
select ST_ChunkBands(rast)
from raster_table;
 st_chunkbands
----------
      3
```

3.8.56. ST_MetaItems

获得栅格对象自定义元数据项的名称列表。

语法

```
text[] ST_metaItems(raster raster_obj);
text[] ST_metaItems(raster raster_obj,
                    integer band);
```

参数

参数名称	描述
raster_obj	需要计算的raster对象。

参数名称	描述
band	波段序号，取值从0开始。

示例

```
-- meta data items of a raster
SELECT ST_MetaItems(raster_obj)
FROM raster_table;
                                                                                st_metaitems

--------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------
 {latitude#long_name,latitude#units,longitude#long_name,longitude#units,NC_GLOBAL#Conventio
ns,NC_GLOBAL#history,NETCDF_DIM_EXTRA,NETCDF_DIM_time_DEF,NETCDF_DIM_time_VALUES}
-- meta data items of a band
SELECT ST_MetaItems(raster_obj, 0)
FROM raster_table;
                                        st_metaitems
--------------------------------------------------------------------------------------------------
{add_offset,long_name,missing_value,NETCDF_DIM_time,NETCDF_VARNAME,scale_factor,units,_Fill
Value}
```

3.8.57. ST_SetMetaData

设置栅格对象或波段的元数据项。

语法

```
raster ST_SetMetaData(raster raster_obj,
                      text key,
                      text value);
raster ST_MetaData(raster raster_obj,
                   integer band,
                   text key,
                   text value);
```

参数

参数名称	描述
raster_obj	需要计算的raster对象。
band	波段序号，取值从0开始。
key	需要设置的元数据项名称。
value	元数据值。

描述

如果传入的元数据值为空值（ `''` ），会删除该元数据项。

示例

```
SELECT ST_MetaData(ST_SetMetaData(rast, 'NETCDF_DIM_time', '12345'), 'NETCDF_DIM_time')
FROM raster_table
st_metadata
------------
12345
SELECT ST_MetaData(ST_SetMetaData(rast, 0, 'NETCDF_DIM_time', '12345'), 0, 'NETCDF_DIM_time
')
FROM raster_table
st_metadata
------------
12345
```

3.8.58. ST_BeginDateTime

获得栅格对象的开始时间。

语法

```
timestamp ST_BeginDateTime(raster raster_obj);
```

参数

参数名称	描述
raster_obj	需要计算的raster对象。

示例

```
SELECT ST_beginDateTime(raster_obj)
FROM raster_table;
    st_begindatetime
-------------------------
Wed Jan 01 00:00:00 2020
```

3.8.59. ST_EndDateTime

获得栅格对象的结束时间。

语法

```
timestamp ST_EndDateTime(raster raster_obj);
```

参数

参数名称	描述
raster_obj	需要计算的raster对象。

示例

```
SELECT ST_endDateTime(raster_obj)
FROM raster_table;
     st_enddatetime
-------------------------
Thu Jan 02 00:00:00 2020
```

3.8.60. ST_SetBeginDateTime

设置栅格对象的开始时间。

语法

```
raster ST_SetBeginDateTime(raster raster_obj,timestamp time);
```

参数

参数名称	描述
raster_obj	需要计算的raster对象。
time	需要设置的时间。建议使用 `yyyy-MM-dd HH:mm:ss` 的格式，例如 `2020-08-30 18:00:00` 。

示例

```
SELECT ST_beginDateTime(ST_setBeginDateTime(raster_obj, '2020-01-01'))
FROM raster_table;
    st_begindatetime
-------------------------
Wed Jan 01 00:00:00 2020
```

3.8.61. ST_SetEndDateTime

设置栅格对象的结束时间。

语法

```
raster ST_SetEndDateTime(raster raster_obj, timestamp time);
```

参数

参数名称	描述
raster_obj	需要计算的raster对象。
time	需要设置的时间。建议使用 `yyyy-MM-dd HH:mm:ss` 的格式，例如 `2020-08-30 18:00:00` 。

示例

```
SELECT ST_endDateTime(ST_setEndDateTime(raster_obj, '2020-01-01'))
FROM raster_table;
    st_enddatetime
-------------------------
Wed Jan 01 00:00:00 2020
```

3.8.62. ST_DateTime

获得栅格对象或波段的时间。

语法

```
text ST_DateTime(raster raster_obj);
timestamp ST_DateTime(raster raster_obj,integer band);
```

参数

参数名称	描述
raster_obj	需要计算的raster对象。
band	波段序号，取值从0开始。

描述

获得栅格对象或波段的时间。如果指定波段，则返回对应波段的时间信息；未指定波段，则返回包含所有波段时间信息的JSON字符串。

示例

```
SELECT ST_DateTime(raster_obj)
FROM raster_table;

st_datetime

----------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------
--------------------------------------
 {"0":"Mon Dec 31 00:00:00 2018","1":"Mon Dec 31 01:00:00 2018","2":"Mon Dec 31 02:00:00 20
18","3":"Mon Dec 31 03:00:00 2018","4":"Mon Dec 31 04:00:00 2018","5":"Mon Dec 31 05:00:00
2018","6":"Mon Dec 31 06:00:00 2018",".
.7":"Mon Dec 31 07:00:00 2018","8":"Mon Dec 31 08:00:00 2018","9":"Mon Dec 31 09:00:00 2018
","10":"Mon Dec 31 10:00:00 2018","11":"Mon Dec 31 11:00:00 2018","12":"Mon Dec 31 12:00:00
2018","13":"Mon Dec 31 13:00:00 2018".
.,"14":"Mon Dec 31 14:00:00 2018","15":"Mon Dec 31 15:00:00 2018","16":"Mon Dec 31 16:00:00
2018","17":"Mon Dec 31 17:00:00 2018","18":"Mon Dec 31 18:00:00 2018","19":"Mon Dec 31 19:0
0:00 2018","20":"Mon Dec 31 20:00:00 .
.2018","21":"Mon Dec 31 21:00:00 2018","22":"Mon Dec 31 22:00:00 2018","23":"Mon Dec 31 23:
00:00 2018"}
SELECT ST_DateTime(raster_obj, 0)
FROM raster_table;
         datetime
--------------------------
"Mon Dec 31 00:00:00 2018"
```

3.8.63. ST_SetDateTime

设置栅格对象的起始和终止时间或波段时间。

语法

```
raster ST_setDateTime(raster raster_obj,
                      timestamp start,
                      timestamp end);
raster ST_setDateTime(raster raster_obj,
                      integer band,
                      timestamp time);
```

参数

参数名称	描述
raster_obj	需要计算的raster对象。
band	波段序号，取值从0开始。
time	波段时间。建议使用 `yyyy-MM-dd HH:mm:ss` 的格式，例如 `2020-08-30 18:00:00` 。
start	开始时间。建议使用 `yyyy-MM-dd HH:mm:ss` 的格式。

参数名称	描述
end	结束时间。建议使用 `yyyy-MM-dd HH:mm:ss` 的格式。

描述

获得栅格对象自定义元数据项的名称列表。

示例

```
select ST_DateTime(ST_setDateTime(raster_obj, 0, '2020-01-02'::timestamp), 0)
FROM raster_table
       st_datetime
-------------------------
 Thu Jan 02 00:00:00 2020
```

3.9. 操作符

3.9.1. 操作符（=）

判断两个raster的ID是否相同。

语法

```
bool  Operator =(raster rast1, raster rast2);
```

参数

参数名称	描述
rast1	raster对象1。
rast2	raster对象2。

说明 仅比较两个raster对象的UUID是否相同，不会对空间范围、像素类型等进行比较。该函数仅用于Union和Btree等操作。

示例

```
SELECT a.rast = b.rast
FROM tbl_a a, tbl_b b
WHERE a.id = b.id
```

3.9.2. 操作符（>）

判断两个raster的ID的比较关系。

语法

```
bool  Operator >(raster rast1, raster rast2);
```

参数

参数名称	描述
rast1	raster对象1。
rast2	raster对象2。

> **说明** 仅比较两个raster对象的UUID是否相同，不会对空间范围、像素类型等进行比较。该函数仅用于Union和Btree等操作。

示例

```
SELECT a.rast > b.rast
FROM tbl_a a, tbl_b b
WHERE a.id = b.id
```

3.9.3. 操作符（<）

判断两个raster的ID的比较关系。

语法

```
bool  Operator <(raster rast1, raster rast2);
```

参数

参数名称	描述
rast1	raster对象1。
rast2	raster对象2。

> **说明** 仅比较两个raster对象的UUID是否相同，不会对空间范围、像素类型等进行比较。该函数仅用于Union和Btree等操作。

示例

```
SELECT a.rast < b.rast
FROM tbl_a a, tbl_b b
WHERE a.id = b.id
```

3.9.4. 操作符（>=）

判断两个raster的ID的比较关系。

语法

```
bool  Operator >=(raster rast1, raster rast2);
```

参数

参数名称	描述
rast1	raster对象1。
rast2	raster对象2。

> **说明** 仅比较两个raster对象的UUID是否相同，不会对空间范围、像素类型等进行比较。该函数仅用于Union和Btree等操作。

示例

```
SELECT a.rast >= b.rast
FROM tbl_a a, tbl_b b
WHERE a.id = b.id
```

3.9.5. 操作符（<=）

判断两个raster的ID的比较关系。

语法

```
bool  Operator <=(raster rast1, raster rast2);
```

参数

参数名称	描述
rast1	raster对象1。
rast2	raster对象2。

> **说明** 仅比较两个raster对象的UUID是否相同，不会对空间范围、像素类型等进行比较。该函数仅用于Union和Btree等操作。

示例

```
SELECT a.rast <= b.rast
FROM tbl_a a, tbl_b b
WHERE a.id = b.id
```

3.10. 空间关系判断

3.10.1. ST_Intersects

判断raster和raster或raster和geometry的空间关系。

语法

```
bool  ST_Intersects(raster rast1, raster rast2);
bool  ST_Intersects(raster rast, geometry geom);
bool  ST_Intersects(geometry geom, raster rast);
```

参数

参数名称	描述
rast1	raster对象1。
rast2	raster对象2。
rast	raster对象。
geom	geometry对象。

示例

```
SELECT a.id
FROM tbl_a a, tbl_b b
WHERE ST_Intersects(a.rast, b.rast)
```

3.10.2. ST_Contains

判断raster和raster或raster和geometry的空间关系。

语法

```
bool  ST_Contains(raster rast1, raster rast2);
bool  ST_Contains(raster rast, geometry geom);
bool  ST_Contains(geometry geom, raster rast);
```

参数

参数名称	描述
rast1	raster对象1。
rast2	raster对象2。
rast	raster对象。
geom	geometry对象。

示例

```
SELECT a.id
FROM tbl_a a, tbl_b b
WHERE ST_Contains(a.rast, b.rast)
```

3.10.3. ST_ContainsProperly

判断raster和raster或raster和geometry的空间关系。

语法

```
bool  ST_ContainsProperly(raster rast1, raster rast2);
bool  ST_ContainsProperly(raster rast, geometry geom);
bool  ST_ContainsProperly(geometry geom, raster rast);
```

参数

参数名称	描述
rast1	raster对象1。
rast2	raster对象2。
rast	raster对象。
geom	geometry对象。

示例

```
SELECT a.id
FROM tbl_a a, tbl_b b
WHERE ST_ContainsProperly(a.rast, b.rast)
```

3.10.4. ST_Covers

判断raster和raster或raster和geometry的空间关系。

语法

```
bool  ST_Covers(raster rast1, raster rast2);
bool  ST_Covers(raster rast, geometry geom);
bool  ST_Covers(geometry geom, raster rast);
```

参数

参数名称	描述
rast1	raster对象1。
rast2	raster对象2。
rast	raster对象。
geom	geometry对象。

示例

```
SELECT a.id
FROM tbl_a a, tbl_b b
WHERE ST_Covers(a.rast, b.rast)
```

3.10.5. ST_CoveredBy

判断raster和raster或raster和geometry的空间关系。

语法

```
bool  ST_CoveredBy(raster rast1, raster rast2);
bool  ST_CoveredBy(raster rast, geometry geom);
bool  ST_CoveredBy(geometry geom, raster rast);
```

参数

参数名称	描述
rast1	raster对象1。
rast2	raster对象2。
rast	raster对象。
geom	geometry对象。

示例

```
SELECT a.id
FROM tbl_a a, tbl_b b
WHERE ST_CoveredBy(a.rast, b.rast)
```

3.10.6. ST_Disjoint

判断raster和raster或raster和geometry的空间关系。

语法

```
bool  ST_Disjoint(raster rast1, raster rast2);
bool  ST_Disjoint(raster rast, geometry geom);
bool  ST_Disjoint(geometry geom, raster rast);
```

参数

参数名称	描述
rast1	raster对象1。
rast2	raster对象2。
rast	raster对象。
geom	geometry对象。

示例

```
SELECT a.id
FROM tbl_a a, tbl_b b
WHERE ST_Disjoint(a.rast, b.rast)
```

3.10.7. ST_overlaps

判断raster和raster或raster和geometry的空间关系。

语法

```
bool  ST_overlaps(raster rast1, raster rast2);
bool  ST_overlaps(raster rast, geometry geom);
bool  ST_overlaps(geometry geom, raster rast);
```

参数

参数名称	描述
rast1	raster对象1。
rast2	raster对象2。
rast	raster对象。
geom	geometry对象。

示例

```
SELECT a.id
FROM tbl_a a, tbl_b b
WHERE ST_Contains(a.rast, b.rast)
```

3.10.8. ST_Touches

判断raster和raster或raster和geometry的空间关系。

语法

```
bool  ST_Touches(raster rast1, raster rast2);
bool  ST_Touches(raster rast, geometry geom);
bool  ST_Touches(geometry geom, raster rast);
```

参数

参数名称	描述
rast1	raster对象1。
rast2	raster对象2。
rast	raster对象。
geom	geometry对象。

示例

```
SELECT a.id
FROM tbl_a a, tbl_b b
WHERE ST_Touches(a.rast, b.rast)
```

3.10.9. ST_Within

判断raster和raster或raster和geometry的空间关系。

语法

```
bool  ST_Within(raster rast1, raster rast2);
bool  ST_Within(raster rast, geometry geom);
bool  ST_Within(geometry geom, raster rast);
```

参数

参数名称	描述
rast1	raster对象1。
rast2	raster对象2。
rast	raster对象。
geom	geometry对象。

示例

```
SELECT a.id
FROM tbl_a a, tbl_b b
WHERE ST_Within(a.rast, b.rast)
```

3.11. 辅助函数

3.11.1. 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();
                                    st_checkgpu
-------------------------------------------------------------------------------------
 [GPU prop]multiProcessorCount=20; sharedMemPerBlock=49152; maxThreadsPerBlock=1024
(1 row)
```

3.11.2. ST_AKId

获取raster对象OSS的AccessKey ID，可用于配合批量修改raster对象的OSS登录密钥。

前提条件

raster对象必须存储在OSS。

语法

```
text ST_AKId(raster raster_obj)
```

参数

参数名称	描述
raster_obj	raster对象。

示例

```
SELECT ST_AKId(rast) from raster_table;
     st_akid
------------------
 OSS_ACCESSKEY_ID
```

3.11.3. ST_SetAccessKey

设置raster对象OSS的登录信息。

前提条件

raster对象必须存储在OSS。

语法

```
raster ST_SetAccessKey(raster raster_obj, text id, text key, bool valid default true)
```

参数

参数名称	描述
raster_obj	raster对象。
id	登录OSS的AccessKey ID，如果为NULL，表示使用之前的登录ID。
key	登录OSS的AccessKey Secret，如果为NULL，表示使用之前的登录Secret。
valid	是否验证登录信息有效性。

示例

```
UPDATE raster_table
SET rast = ST_SetAccessKey(rast, 'OSS_ACCESSKEY_ID', 'OSS_ACCESSKEY_SECRET');
SELECT ST_AKId(rast) from raster_table;
     st_akid
------------------
 OSS_ACCESSKEY_ID
```

3.11.4. ST_SetAKId

设置raster对象OSS的AccessKey ID。

前提条件

raster对象必须存储在OSS。

语法

```
raster ST_SetAKId(raster raster_obj, text id, bool valid default true)
```

参数

参数名称	描述
raster_obj	raster对象。
id	登录OSS的AccessKey ID，如果为NULL，表示使用之前的登录ID。
valid	是否验证登录信息有效性。

示例

```
UPDATE raster_table
SET rast = ST_SetAKId(rast, 'OSS_ACCESSKEY_ID');
SELECT ST_AKId(rast) from raster_table;
     st_akid
------------------
 OSS_ACCESSKEY_ID
```

3.11.5. ST_SetAKSecret

设置raster对象OSS的AccessKey Secret。

前提条件

raster对象必须存储在OSS。

语法

```
raster ST_SetAKSecret(raster raster_obj, text key, bool valid default true)
```

参数

参数名称	描述
raster_obj	raster对象。
key	登录OSS的AccessKey Secret，如果为NULL，表示使用之前的登录Secret。
valid	是否验证登录信息有效性。

示例

```
UPDATE raster_table
SET rast = ST_SetAKSecret(rast, 'OSS_ACCESSKEY_SECRET');
```

3.12. 变量

3.12.1. ganos.raster.calculate_md5

指定是否在导入raster对象时计算MD5值并记录到元数据中。

类型

boolean

默认值

false

> 说明　取值：true|false。

描述

指定是否在导入raster对象时计算MD5值并记录到元数据中。

示例

```
Set ganos.raster.calculate_md5 = true;
```

3.12.2. ganos.raster.md5sum_chunk_size

指定计算MD5时每次读入的缓存的大小，单位是MB。

类型

integer

默认值

10

> 说明　取值范围：1~256。

示例

```
Set ganos.raster.md5sum_chunk_size = 20;
```

3.12.3. ganos.raster.mosaic_must_same_nodata

指定镶嵌时数据源的NoData值是否必须相同。镶嵌时并不会对NoData值进行转换，如果该变量的取值为false时可能会导致镶嵌后的像素语义不一致。

类型

boolean

默认值

true

> **说明** 取值：true|false。

示例

```
Set ganos.raster.mosaic_must_same_nodata = false;
```

4.SpatialRef SQL参考

4.1. ST_SrEqual

判断两个空间参考是否相同。

语法

```
boolean ST_SrEqual(cstring sr1, cstring sr2, boolean strict default true);
```

参数

参数名称	描述
sr1	空间参考1的字符串。必须是OGC WKT或者Proj4形式的字符串。
sr2	空间参考2的字符串。必须是OGC WKT或者Proj4形式的字符串。
strict	是否采用严格比较方式。如为true，则会对参考椭球体的名称进行比较。默认为true。

描述

本函数通过解析语义的方式对空间参考进行比较，会对投影方式、参考椭球、长短半轴等参数信息进行比较。如果相同，返回t；如果不同，返回f。

示例

```
--比较两个基于文本的空间参考。
select ST_srEqual('GEOGCS["WGS 84",DATUM["WGS_1984",SPHEROID["WGS 84",6378137,298.257223563
,AUTHORITY["EPSG","7030"]],AUTHORITY["EPSG","6326"]],PRIMEM["Greenwich",0,AUTHORITY["EPSG",
"8901"]],UNIT["degree",0.0174532925199433,AUTHORITY["EPSG","9122"]],AUTHORITY["EPSG","4326"
]]', '+proj=longlat +datum=WGS84 +no_defs');
st_srequal
------------
 t
 --寻找spatial_ref_sys表中空间参考的srid。
 select srid from spatial_ref_sys where st_srequal(srtext::cstring, '+proj=longlat +ellps=G
RS80 +no_defs ') limit 1;
 srid
------
 3824
```

4.2. ST_SrReg

注册一个新的空间参考。

语法

```
integer ST_SrReg(cstring sr);
integer ST_SrReg(cstring auth_name, integer auth_id, cstring sr);
```

参数

参数名称	描述
sr	空间参考字符串，必须是OGC WKT或者Proj4形式的字符串。
auth_name	空间参考系统定义的作者，例如EPSG。
auth_id	空间参考系统定义的空间参考ID。

描述

如果空间参考已经存在，则返回已经存在的空间参考srid；如果空间参考不存在，则会向spatial_ref_sys表中插入一条记录并返回新空间参考的srid。

示例

```
--空间参考已存在
select 4490, ST_srReg('GEOGCS["China Geodetic Coordinate System 2000",DATUM["China_2000",SP
HEROID["CGCS2000",6378137,298.257222101,AUTHORITY["EPSG","1024"]],AUTHORITY["EPSG","1043"]]
,PRIMEM["Greenwich",0,AUTHORITY["EPSG","8901"]],UNIT["degree",0.0174532925199433,AUTHORITY[
"EPSG","9122"]],AUTHORITY["EPSG","4490"]]');
 st_srreg
----------
     4490
--新空间参考
select ST_srReg('user_defined',100, 'GEOGCS["User Geodetic Coordinate System ",DATUM["China
_2000",SPHEROID["CGCS2000",6378137,298.257222101,AUTHORITY["EPSG","903"]],AUTHORITY["EPSG",
"1043"]],PRIMEM["Greenwich",0,AUTHORITY["EPSG","8901"]],UNIT["degree",0.0174532925199433,AU
THORITY["EPSG","9122"]],AUTHORITY["EPSG","4491"]]');
 st_srreg
----------
    10001
select ST_srReg('+proj=tmerc +lat_0=1 +lon_0=112 +k=1 +x_0=19500001 +y_0=0 +ellps=krass +to
wgs84=24.47,-130.89,-81.56,0,0,0.13,-0.22 +units=m +no_defs');
 st_srreg
----------
   10002
```

4.3. ST_SrFromEsriWkt

将一个Esri Wkt规范的字符串转为OGC规范的字符串。

语法

```
cstring ST_SrFromEsriWkt(cstring sr);
```

参数

参数名称	描述
sr1	基于Esri Wkt规范的空间参考字符串。

示例

```
SELECT ST_srFromEsriWkt('GEOGCS["China Geodetic Coordinate System 2000",DATUM["D_China_2000
",SPHEROID["CGCS2000",6378137,298.257222101]],PRIMEM["Greenwich",0],UNIT["Degree",0.0174532
92519943295]]');
st_srfromesriwkt

----------------------------------------------------------------------------------------------
----------------------------------------------------------------------------
 GEOGCS["China Geodetic Coordinate System 2000",DATUM["China_2000",SPHEROID["CGCS2000",6378
137,298.257222101]],PRIMEM["Greenwich",0],UNIT["Degree",0.017453292519943295]]
```

5.Trajectory SQL参考

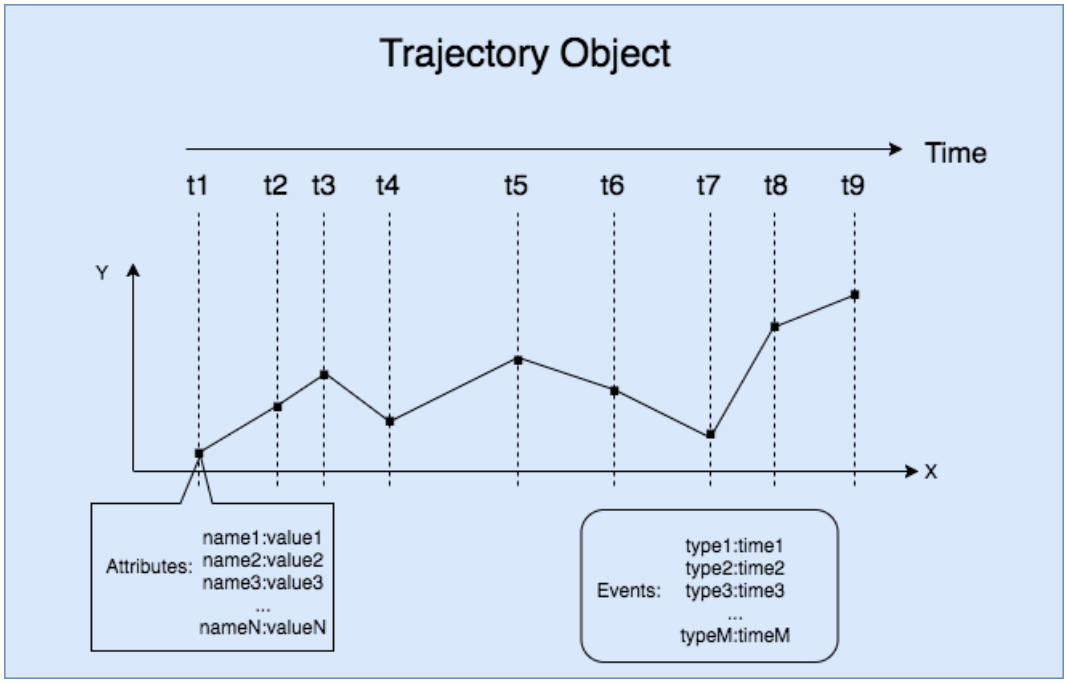
5.1. 基本概念

本章节将为您介绍Trajectory SQL的基本概念。

Trajectory对象

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。

轨迹示意图如下。



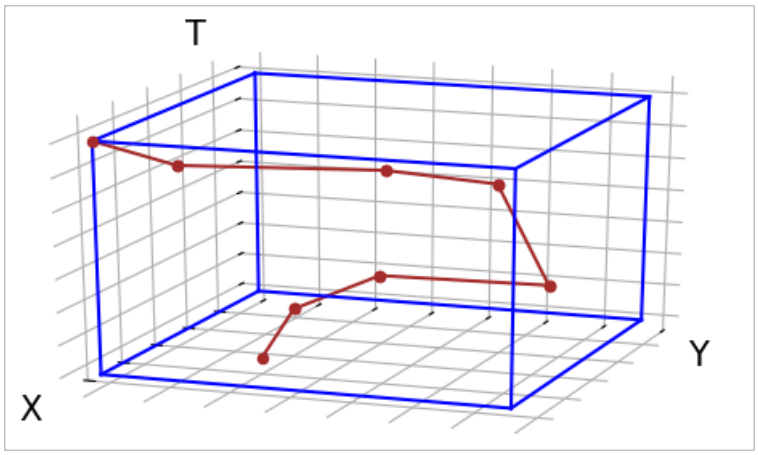
BoxNdf对象

在实际计算时，由于轨迹模块涉及的对象（轨迹类型或几何类型）常常十分复杂，在分析时可能会使用计算较为简单的矩形框对象描述查询或简化计算。在轨迹模块，我们使用BoxNdf类型表示矩形框。

BoxNdf对象表示在时空中的一个多维立方体区域，其由在 `x,y,z（空间）,t（时间）` 四个坐标轴上的最小值和最大值表示。每个矩形框包含的维度可能不同，有的矩形框只记录x、y维度信息，有些则可能记录x、y、z、t四个维度的信息。

在查询时可能用矩形框表示一个查询范围，在查询时也可以利用轨迹或几何类型的外包矩形框来辅助查询。例如下图就是一条在 `x,y,t` 空间中的运动的轨迹和它对应的外包矩形框。

外包矩形框示意图如下。



5.2. 构造函数

5.2.1. 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。

5.2.2. 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 , anyarrayattr_any );
```

参数

参数名称	描述
type	轨迹的类型，目前只支持 ST_POINT。
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,"attribut
es":{"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]}, "ve
sname": {"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 14:30:0
0","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":[120,130,14
0]},"accuracy":{"type":"float","length":4,"nullable":false,"value":[120.0,130.0,140.0]},"be
aring":{"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 15:00:0
0"},{"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": [1
20, 130, 140]}, "accuracy": {"type": "float", "length": 4, "nullable" : false,"value": [120
, 130, 140]}, "bearing": {"type": "float", "length": 8, "nullable" : false,"value": [120, 1
30, 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 14:30:0
0","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":[120,130,14
0]},"accuracy":{"type":"float","length":4,"nullable":false,"value":[120.0,130.0,140.0]},"be
aring":{"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 15:00:0
0"},{"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", "le
ngth": 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", "nu
llable" : 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_maketraject
ory
----------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------
---------------------------------
 {"trajectory":{"version":1,"type":"STPOINT","leafcount":3,"start_time":"2010-01-01 14:30:0
0","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":[120,130,14
0]},"accuracy":{"type":"float","length":4,"nullable":false,"value":[120.0,130.0,140.0]},"be
```

```
aring":{"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 15:00:0
0"},{"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, ARR
AY['2010-01-01 11:30'::timestamp], ARRAY['velocity'], ARRAY[1::int4], NULL, NULL, NULL::any
array);
                                                        st_maketrajectory

------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
{"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":t
rue,"value":[1]}}}}
(1 row)
```

5.2.3. 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_attr
s_json) ;
trajectory ST_append(trajectory traj, trajectory 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。

参数名称	描述
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": {"typ
e": "float", "length": 8, "nullable" : false,"value": [120, 130, 140]}, "acceleration": {"t
ype": "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_makeTr
ajectory('STPOINT', 'LINESTRING(7 7, 3 4, 1 5)'::geometry, '[2010-01-02 15:30, 2010-01-02 1
8:00)'::tsrange,  '{"leafcount":3,"attributes":{"velocity": {"type": "integer", "length": 2
,"nullable" : true,"value": [121, 131, 141]}, "accuracy": {"type": "float", "length": 4, "n
ullable" : false,"value": [121, 131, 141]}, "bearing": {"type": "float", "length": 8, "null
able" : 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 11:30:0
0","end_time":"2010-01-02 18:00:00","spatial":"LINESTRING(1 1,6 6,9 8,7 7,3 4,1 5)","timeli
ne":["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":{"typ
e":"integer","length":2,"nullable":true,"value":[120,130,140,121,131,141]},"accuracy":{"typ
e":"float","length":4,"nullable":false,"value":[120.0,130.0,140.0,121.0,131.0,141.0]},"bear
ing":{"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":["201
0-01-01 14:30:00","2010-01-01 15:00:00","2010-01-01 15:30:00","2010-01-02 14:30:00","2010-0
1-02 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)
```

5.3. 编辑与处理函数

5.3.1. 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, c
string 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,-1
79.42078 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 51.8
4791,-179.38467 51.85114,-179.38216 51.85439,-179.37997 51.85762,-179.37772 51.86144,-179.3
7474 51.86568,-179.37219 51.86869,-179.36983 51.87156,-179.36755 51.87467,-179.36001 51.884
23,-179.35754 51.88712,-179.34216 51.90644,-179.33935 51.90995,-179.33704 51.91298,-179.188
26 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,-1
```

```
52.90001,-177.47319 52.90084,-177.46251 52.90268,-177.38188 52.91595,-177.37102 52.91765,-1
77.36378 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 53.0
0060,-176.86594 53.00256,-176.85946 53.00370,-176.85294 53.00481,-176.84640 53.00592,-176.8
3985 53.00705,-176.83238 53.00830,-176.82589 53.00950,-176.81848 53.01084,-176.80553 53.013
10,-176.79879 53.01419,-176.79115 53.01548,-176.78466 53.01668,-176.77901 53.01765,-176.772
56 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,'201
7-01-15 09:12:29'::timestamp,'2017-01-15 09:13:39'::timestamp,'2017-01-15 09:14:48'::timest
amp,'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 0
9:34:40'::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'::timesta
mp,'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'::timestam
p,'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,'2017-0
1-15 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'::t
imestamp,'2017-01-15 11:36:58'::timestamp,'2017-01-15 11:42:00'::timestamp,'2017-01-15 11:4
3:10'::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'::ti
mestamp,'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'::tim
estamp,'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,'2017-01-1
5 20:47:29'::timestamp,'2017-01-15 20:48:40'::timestamp,'2017-01-15 20:49:49'::timestamp,'2
017-01-15 20:50:59'::timestamp,'2017-01-15 20:52:21'::timestamp,'2017-01-15 20:53:29'::time
stamp,'2017-01-15 20:54:50'::timestamp,'2017-01-15 20:57:09'::timestamp,'2017-01-15 20:58:2
0'::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,'201
7-01-15 21:05:50'::timestamp,'2017-01-15 21:07:29'::timestamp,'2017-01-15 21:09:11'::timest
amp,'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,"nullabl
e":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,
74.2,72.2,80.5,78.6,77.3,70.9,80.1,85.4,71.9,67.0,77.5,77.5,72.2,70.5,72.6,70.8,77.8,71.2,7
1.2,73.8,75.4,67.1,77.5,74.3,76.9,80.1,72.8,76.0,75.4,72.9]},"heading" : {"type":"float","l
ength":8,"nullable":false,"value":[22.0,23.0,23.0,23.0,23.0,21.0,21.0,25.0,24.0,26.0,25.0,2
7.0,28.0,29.0,31.0,30.0,28.0,29.0,29.0,28.0,28.0,27.0,24.0,24.0,25.0,25.0,25.0,26.0,25.0,24
```

```
.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 09:06:3
9","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 09:34:4
0","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":"flo
at","length":8,"nullable":false,"value":[10.5,11.0,10.6,11.2,10.1,9.9,12.3,12.7]},"cog":{"t
ype":"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":"2
017-01-15 21:18:30","spatial":"LINESTRING(-179.48077 51.72814,-179.42595 51.80094,-179.3973
4 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 0
9:47:38","2017-01-15 09:59:09","2017-01-15 11:36:58","2017-01-15 11:50:28","2017-01-15 18:0
1:00","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 09:06:3
9","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 52.9
0084,-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":{"leafc
ount":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":{"ty
pe":"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 09:06:3
9","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","201
7-01-15 10:17:29","2017-01-15 11:30:09","2017-01-15 11:36:58","2017-01-15 11:50:28","2017-0
1-15 18:01:00","2017-01-15 21:18:30"],"attributes":{"leafcount":10,"sog":{"type":"float","l
ength":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)
```

5.3.2. 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 09:06:3
9","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 52.9
7394,-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)
```

5.3.3. 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 51.81
941,-179.40751 51.82223,-179.40497 51.82505,-179.40242 51.82796,-179.39981 51.83095,-179.39
734 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" : {"he
ading" : {"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 09:17:4
8","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 0
9:17:48","2017-01-15 09:23:19","2017-01-15 09:38:09","2017-01-15 09:39:18","2017-01-15 21:1
8:3
0"],"attributes":{"leafcount":6,"heading":{"type":"float","length":4,"nullable":false,"valu
e":[23.0,21.0,72.0,73.0,74.0,73.0]}}}}
(1 row)
```

5.3.4. 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 51.76
273,-179.44845 51.77186,-179.43419 51.78977,-179.41259 51.81643,-179.41001 51.81941,-179.40
751 51.82223,-179.40497 51.82505,-179.40242 51.82796,-179.39981 51.83095,-179.39734 51.8339
8,-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'::tim
estamp,'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,'2017-01-1
5 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 5
1.7
4934,-179.46731 51.74634,-179.46183 51.75378,-179.4556 51.76273,-179.45943 51.75736,-179.44
845 51.77186,-179.41259 51.81643,-179.43419 51.78977,-179.41001 51.81941,-179.40751 51.8222
3,-
179.40497 51.82505,-179.40242 51.82796,-179.39981 51.83095,-179.39499 51.83709,-179.39734 5
1.83398)","timeline":["2017-01-15 09:06:39","2017-01-15 09:13:39","2017-01-15 09:14:48","20
17-
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)
```

5.3.5. ST_deviation

计算处理后的轨迹与原始轨迹之间的偏差。

语法

```
float8 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)
```

5.4. 属性元数据

5.4.1. 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":"2010-01-0
1 11:30:00","end_time":"2010-01-01 12:30:00","spatial":"SRID=4326;LINESTRING(1 1,3 5)","tim
eline":["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","le
ngth":8,"nullable":true,"value":[null,1.0]},"angel":{"type":"string","length":64,"nullable"
:true,"value":["test",null]}, "tngel2":{"type":"timestamp","length":8,"nullable":true,"valu
e":["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,"nulla
ble":true}}
(1 row)
```

5.4.2. 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":"2010-01-0
1 11:30:00","end_time":"2010-01-01 12:30:00","spatial":"SRID=4326;LINESTRING(1 1,3 5)","tim
eline":["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","le
ngth":8,"nullable":true,"value":[null,1.0]},"angel":{"type":"string","length":64,"nullable"
:true,"value":["test",null]}, "tngel2":{"type":"timestamp","length":8,"nullable":true,"valu
e":["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)
```

5.4.3. 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":"2010-01-0
1 11:30:00","end_time":"2010-01-01 12:30:00","spatial":"SRID=4326;LINESTRING(1 1,3 5)","tim
eline":["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","le
ngth":8,"nullable":true,"value":[null,1.0]},"angel":{"type":"string","length":64,"nullable"
:true,"value":["test",null]}, "tngel2":{"type":"timestamp","length":8,"nullable":true,"valu
e":["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":"2010-01-0
1 11:30:00","end_time":"2010-01-01 12:30:00","spatial":"SRID=4326;LINESTRING(1 1,3 5)","tim
eline":["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","le
ngth":8,"nullable":true,"value":[null,1.0]},"angel":{"type":"string","length":64,"nullable"
:true,"value":["test",null]}, "tngel2":{"type":"timestamp","length":8,"nullable":true,"valu
e":["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)
```

5.4.4. 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":"2010-01-0
1 11:30:00","end_time":"2010-01-01 12:30:00","spatial":"SRID=4326;LINESTRING(1 1,3 5)","tim
eline":["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","le
ngth":8,"nullable":true,"value":[null,1.0]},"angel":{"type":"string","length":64,"nullable"
:true,"value":["test",null]}, "tngel2":{"type":"timestamp","length":8,"nullable":true,"valu
e":["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":"2010-01-0
1 11:30:00","end_time":"2010-01-01 12:30:00","spatial":"SRID=4326;LINESTRING(1 1,3 5)","tim
eline":["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","le
ngth":8,"nullable":true,"value":[null,1.0]},"angel":{"type":"string","length":64,"nullable"
:true,"value":["test",null]}, "tngel2":{"type":"timestamp","length":8,"nullable":true,"valu
e":["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)
```

5.4.5. 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":"2010-01-0
1 11:30:00","end_time":"2010-01-01 12:30:00","spatial":"SRID=4326;LINESTRING(1 1,3 5)","tim
eline":["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","le
ngth":8,"nullable":true,"value":[null,1.0]},"angel":{"type":"string","length":64,"nullable"
:true,"value":["test",null]}, "tngel2":{"type":"timestamp","length":8,"nullable":true,"valu
e":["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":"2010-01-0
1 11:30:00","end_time":"2010-01-01 12:30:00","spatial":"SRID=4326;LINESTRING(1 1,3 5)","tim
eline":["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","le
ngth":8,"nullable":true,"value":[null,1.0]},"angel":{"type":"string","length":64,"nullable"
:true,"value":["test",null]}, "tngel2":{"type":"timestamp","length":8,"nullable":true,"valu
e":["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)
```

5.4.6. 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":"2010-01-0
1 11:30:00","end_time":"2010-01-01 12:30:00","spatial":"SRID=4326;LINESTRING(1 1,3 5)","tim
eline":["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","le
ngth":8,"nullable":true,"value":[null,1.0]},"angel":{"type":"string","length":64,"nullable"
:true,"value":["test",null]}, "tngel2":{"type":"timestamp","length":8,"nullable":true,"valu
e":["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":"2010-01-0
1 11:30:00","end_time":"2010-01-01 12:30:00","spatial":"SRID=4326;LINESTRING(1 1,3 5)","tim
eline":["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","le
ngth":8,"nullable":true,"value":[null,1.0]},"angel":{"type":"string","length":64,"nullable"
:true,"value":["test",null]}, "tngel2":{"type":"timestamp","length":8,"nullable":true,"valu
e":["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)
```

5.5. 事件函数

5.5.1. 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":"2010-01-0
1 11:30:00","end_time":"2010-01-01 12:30:00","spatial":"SRID=4326;LINESTRING(1 1,3 5)","tim
eline":["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","le
ngth":8,"nullable":true,"value":[null,1.0]},"angel":{"type":"string","length":64,"nullable"
:true,"value":["test",null]}, "tngel2":{"type":"timestamp","length":8,"nullable":true,"valu
e":["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 11:30:0
0","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,"va
lue":["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,tr
ue]}},"events":[{"1":"2010-01-01 11:30:00"}]}}
(1 row)
```

5.5.2. 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":"2010-01-0
1 11:30:00","end_time":"2010-01-01 12:30:00","spatial":"SRID=4326;LINESTRING(1 1,3 5)","tim
eline":["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","le
ngth":8,"nullable":true,"value":[null,1.0]},"angel":{"type":"string","length":64,"nullable"
:true,"value":["test",null]}, "tngel2":{"type":"timestamp","length":8,"nullable":true,"valu
e":["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)
```

5.5.3. 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。
index	事件索引序号。

示例

```
With traj AS (
  select '{"trajectory":{"version":1,"type":"STPOINT","leafcount":2,"start_time":"2010-01-0
1 11:30:00","end_time":"2010-01-01 12:30:00","spatial":"SRID=4326;LINESTRING(1 1,3 5)","tim
eline":["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","le
ngth":8,"nullable":true,"value":[null,1.0]},"angel":{"type":"string","length":64,"nullable"
:true,"value":["test",null]}, "tngel2":{"type":"timestamp","length":8,"nullable":true,"valu
e":["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)
```

5.5.4. 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":"2010-01-0
1 11:30:00","end_time":"2010-01-01 12:30:00","spatial":"SRID=4326;LINESTRING(1 1,3 5)","tim
eline":["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","le
ngth":8,"nullable":true,"value":[null,1.0]},"angel":{"type":"string","length":64,"nullable"
:true,"value":["test",null]}, "tngel2":{"type":"timestamp","length":8,"nullable":true,"valu
e":["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}
```

5.5.5. 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":"2010-01-0
1 11:30:00","end_time":"2010-01-01 12:30:00","spatial":"SRID=4326;LINESTRING(1 1,3 5)","tim
eline":["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","le
ngth":8,"nullable":true,"value":[null,1.0]},"angel":{"type":"string","length":64,"nullable"
:true,"value":["test",null]}, "tngel2":{"type":"timestamp","length":8,"nullable":true,"valu
e":["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
```

5.6. 属性函数

5.6.1. 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;
```

5.6.2. 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;
```

5.6.3. 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 ST_AsText(ST_trajectorySpatial(traj)) FROM traj_table;
            st_astext
---------------------------------
 LINESTRING(114 35,115 36,116 37)
```

5.6.4. 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('LINEST
RING (114 35, 115 36, 116 37)', 4326), '[2010-01-01 14:30, 2010-01-01 15:30)'::tsrange, nul
l));
                              st_trajectorytemporal
-------------------------------------------------------------------------------
 {"timeline":["2010-01-01 14:30:00","2010-01-01 15:00:00","2010-01-01 15:30:00"]}
(1 row)
```

5.6.5. 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;
```

5.6.6. 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,-1
79.40751 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
```

5.6.7. 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,-1
79.40751 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
```

5.6.8. 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 51.7
2814,-179.46731 51.74634,-179.46502 51.74934,-179.46183 51.75378,-179.45943 51.75736,-179.4
5560 51.76273,-179.44845 51.77186,-179.43419 51.78977,-179.41259 51.81643,-179.41001 51.819
41,-179.40751 51.82223,-179.40497 51.82505,-179.40242 51.82796,-179.39981 51.83095,-179.397
34 51.83398,-179.39499 51.83709)'::geometry, ARRAY['2017-01-15 09:06:39'::timestamp,'2017-0
1-15 09:14:48','2017-01-15 09:13:39','2017-01-15 09:16:28','2017-01-15 09:19:48','2017-01-1
5 09:17:48','2017-01-15 09:23:19','2017-01-15 09:34:40','2017-01-15 09:30:28','2017-01-15 0
9:36:59','2017-01-15 09:38:09','2017-01-15 09:39:18','2017-01-15 09:40:40','2017-01-15 09:4
7:38','2017-01-15 21:18:30','2017-01-15 09:48:49'], '{"leafcount": 16, "attributes" : {"hea
ding" : {"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
```

5.6.9. 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 51.728
14,-179.46731 51.74634,-179.46502 51.74934,-179.46183 51.75378,-179.45943 51.75736,-179.455
60 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,'2017-01-1
5 09:13:39','2017-01-15 09:14:48','2017-01-15 09:16:28','2017-01-15 09:17:48','2017-01-15 0
9:19:48','2017-01-15 09:23:19','2017-01-15 09:30:28','2017-01-15 09:34:40','2017-01-15 09:3
6:59','2017-01-15 09:38:09','2017-01-15 09:39:18','2017-01-15 09:40:40','2017-01-15 09:47:3
8','2017-01-15 09:48:49','2017-01-15 21:18:30'], '{"leafcount": 16, "attributes" : {"headin
g" : {"type": "float", "length": 4, "nullable" : false,"value": [23.0,23.0,23.0,23.0,21.0,2
1.0,72.0,72.0,72.0,72.0,73.0,74.0,73.0,73.0,73.0,73.0]}}}' ),'heading');
 st_attrfloatmax
-----------------
              74
```

5.6.10. 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 51.728
14,-179.46731 51.74634,-179.46502 51.74934,-179.46183 51.75378,-179.45943 51.75736,-179.455
60 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,'2017-01-1
5 09:13:39','2017-01-15 09:14:48','2017-01-15 09:16:28','2017-01-15 09:17:48','2017-01-15 0
9:19:48','2017-01-15 09:23:19','2017-01-15 09:30:28','2017-01-15 09:34:40','2017-01-15 09:3
6:59','2017-01-15 09:38:09','2017-01-15 09:39:18','2017-01-15 09:40:40','2017-01-15 09:47:3
8','2017-01-15 09:48:49','2017-01-15 21:18:30'], '{"leafcount": 16, "attributes" : {"headin
g" : {"type": "float", "length": 4, "nullable" : false,"value": [23.0,23.0,23.0,23.0,21.0,2
1.0,72.0,72.0,72.0,72.0,73.0,74.0,73.0,73.0,73.0,73.0]}}}' ),'heading');
 st_attrfloatmin
-----------------
              21
```

5.6.11. 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 51.8
1941,-179.40751 51.82223,-179.40497 51.82505,-179.40242 51.82796,-179.39981 51.83095,-179.3
9734 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" : {"he
ading" : {"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)
```

5.6.12. 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)
```

5.6.13. 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
```

5.6.14. 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)
```

5.6.15. 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, geometry g);
```

参数

参数名称	描述
traj	轨迹对象。
g	点位置。

示例

```
Select ST_timeAtPoint(traj, st_geomfromtext('POINT(115 36)')) from traj_table;
     st_timeatpoint
-------------------------
 {"2010-01-01 15:00:00"}
```

5.6.16. 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;
```

5.6.17. 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

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

5.6.18. 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,'2010-1-11 23:40:00') From traj_table;
```

5.6.19. 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;
```

5.6.20. 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;
```

5.6.21. 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;
```

5.6.22. 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;
```

5.6.23. 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。
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;
```

5.6.24. 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;
```

5.6.25. ST_samplingInterval

获取采样间隔。

语法

```
interval 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)
```

5.6.26. 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":"2010-01-0
1 11:30:00","end_time":"2010-01-01 12:30:00","spatial":"SRID=4326;LINESTRING(1 1,3 5)","tim
eline":["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","le
ngth":8,"nullable":true,"value":[null,1.0]},"angel":{"type":"string","length":64,"nullable"
:true,"value":["test",null]}, "tngel2":{"type":"timestamp","length":8,"nullable":true,"valu
e":["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":"2010-01-0
1 11:30:00","end_time":"2010-01-01 12:30:00","spatial":"SRID=4326;LINESTRING(1 1,3 5)","tim
eline":["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","le
ngth":8,"nullable":true,"value":[null,1.0]},"angel":{"type":"string","length":64,"nullable"
:true,"value":["test",null]}, "tngel2":{"type":"timestamp","length":8,"nullable":true,"valu
e":["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":"2010-01-0
1 11:30:00","end_time":"2010-01-01 12:30:00","spatial":"SRID=4326;LINESTRING(1 1,3 5)","tim
eline":["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","le
ngth":8,"nullable":true,"value":[null,1.0]},"angel":{"type":"string","length":64,"nullable"
:true,"value":["test",null]}, "tngel2":{"type":"timestamp","length":8,"nullable":true,"valu
e":["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)
```

5.6.27. 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":"2010-01-0
1 11:30:00","end_time":"2010-01-01 12:30:00","spatial":"SRID=4326;LINESTRING(1 1,3 5)","tim
eline":["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","le
ngth":8,"nullable":true,"value":[null,1.0]},"angel":{"type":"string","length":64,"nullable"
:true,"value":["test",null]}, "tngel2":{"type":"timestamp","length":8,"nullable":true,"valu
e":["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)
```

5.6.28. 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":"2010-01-0
1 11:30:00","end_time":"2010-01-01 12:30:00","spatial":"SRID=4326;LINESTRING(1 1,3 5)","tim
eline":["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","le
ngth":8,"nullable":true,"value":[null,1.0]},"angel":{"type":"string","length":64,"nullable"
:true,"value":["test",null]}, "tngel2":{"type":"timestamp","length":8,"nullable":true,"valu
e":["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)
```

5.6.29. 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":"2010-01-0
1 11:30:00","end_time":"2010-01-01 12:30:00","spatial":"SRID=4326;LINESTRING(1 1,3 5)","tim
eline":["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","le
ngth":8,"nullable":true,"value":[null,1.0]},"angel":{"type":"string","length":64,"nullable"
:true,"value":["test",null]}, "tngel2":{"type":"timestamp","length":8,"nullable":true,"valu
e":["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)
```

5.6.30. 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":"2010-01-0
1 11:30:00","end_time":"2010-01-01 12:30:00","spatial":"SRID=4326;LINESTRING(1 1,3 5)","tim
eline":["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","le
ngth":8,"nullable":true,"value":[null,1.0]},"angel":{"type":"string","length":64,"nullable"
:true,"value":["test",null]}, "tngel2":{"type":"timestamp","length":8,"nullable":true,"valu
e":["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}
```

5.6.31. 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 val
ue);
int8[] ST_attrIntFilter(trajectory traj, cstring attr_field_name,cstring operator, int8 val
ue1, int8 value2);
```

参数

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

参数	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(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,-1
79.45560 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,'20
17-01-15 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)
```

5.6.32. 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, floa
t8 value);
float8[] ST_attrFloatFilter(trajectory traj, cstring attr_field_name,cstring operator, floa
t8 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。
operator	过滤符，`'='`，`'!='`，`'>'`，`'<'`，`'>='`，`'<='`，`'[]'`，`'(]'`，`'[)'`，`'()'`。
value、value1	属性固定值下限。

参数名称	描述
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)
```

5.6.33. 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。
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。

示例

```
insert into traj values(3,ST_makeTrajectory('STPOINT'::leaftype, st_geomfromtext('LINESTRIN
G (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 Ja
n 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)
```

5.6.34. 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)
```

5.6.35. 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 5
1.72814,-179.46731 51.74634,-179.46502 51.74934,-179.46183 51.75378,-179.45943 51.75736,-17
9.45560 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,'201
7-01-15 09:13:39','2017-01-15 09:14:48','2017-01-15 09:16:28','2017-01-15 09:17:48','2017-0
1-15 09:19:48','2017-01-15 09:23:19','2017-01-15 09:30:28','2017-01-15 09:34:40','2017-01-1
5 09:36:59','2017-01-15 09:38:09','2017-01-15 09:39:18','2017-01-15 09:40:40','2017-01-15 0
9:47:38','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,2
1.0,21.0,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 5
1.7
4634,-179.46502 51.74934,-179.45943 51.75736,-179.4556 51.76273,-179.43419 51.78977,-179.41
259 51.81643,-179.40751 51.82223,-179.40497 51.82505,-179.40242 51.82796,-179.39981 51.8309
5,-
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,"nu
lla
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)
```

5.6.36. 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

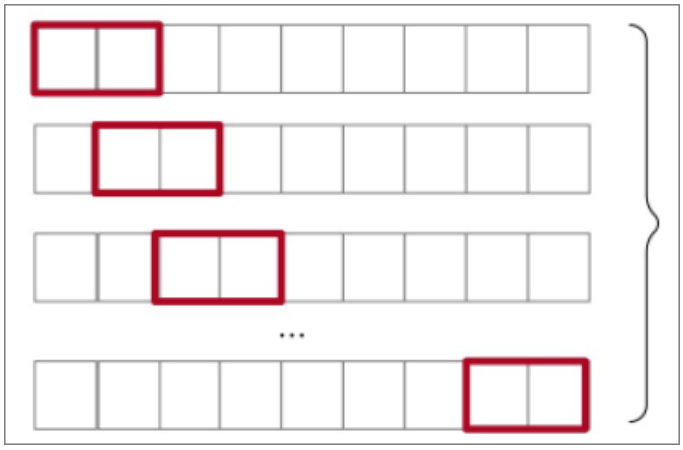
```
SETOF record ST_trajAttrsMeanMax(trajectory traj, cstring attr_field_name, out interval dur
ation, out float8 max);
```

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', '2010-01-0
1 14:30'],
    '{"leafcount":4, "attributes":{"velocity": {"type": "float", "length": 8,"nullable" : t
rue,"value": [120.0, 130.0, 140.0, 120.0]}, "power": {"type": "float", "length": 4,"nullabl
e" : 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', '2010-01-0
1 14:30'],
    '{"leafcount":4, "attributes":{"velocity": {"type": "float", "length": 8,"nullable" : t
rue,"value": [120.0, 130.0, 140.0, 120.0]}, "power": {"type": "float", "length": 4,"nullabl
e" : 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)
```

5.7. 外包框类型和处理函数

5.7.1. ST_MakeBox

获取指定类型在指定时间段内的外包框。

语法

```
boxndf ST_MakeBox(geometry geom);
boxndf ST_MakeBox(trajectory traj);
boxndf ST_MakeBox(geometry geom, timestamp ts, timestamp te);
boxndf ST_MakeBox(trajectory traj, timestamp ts, timestamp te);
boxndf ST_MakeBox(timestamp ts, timestamp te);
```

参数

参数名称	描述
geom	需要获取的外包框的几何对象。
traj	需要获取的外包框的轨迹对象。
ts	指定时间段的开始时间。
te	指定时间段的结束时间。

描述

外包框即包含某个时空物体的最小多维矩形。

- 对于二维或三维的空间对象，返回其对应维度的外包框。
- 对于指定空间对象和时间段起止时间的调用 `ST_MakeBox(geometry geom, timestamp ts, timestamp te)`，将返回目标对象在目标时间段内的外包框。
- 对于轨迹，若无时间段参数，则返回其整体的外包框；若有时间段参数，则返回在指定时间段内子轨迹的外包框。
- 由于BoxNDF内部由float类型表示，因此得到的Box可能会略大于输入的参数，例如下界比实际值略小或上界比实际值略大。

示例

```
WITH geom AS(
    SELECT ('POLYGON((12.7243236691148 4.35238368367118,12.9102992732078 1.49748113937676,12.5926592946053 1.67643963359296' ||
            ',12.0197574747333 3.19258554889152,12.7243236691148 4.35238368367118))')::geometry a
)
SELECT ST_MakeBox(a),ST_MakeBox(a,'2000-01-01 00:00:10'::timestamp, '2000-01-01 02:13:20'::timestamp) from geom;
                          st_makebox                          |                          st_makebox
--------------------------------------------------------------+--------------------------------------------------------------
 BOX2D(12.0197572708 1.49748110771,12.9102993011 4.35238409042) | BOX2DT(12.0197572708 1.49748110771 2000-01-01 00:00:09.999999,12.9102993011 4.35238409042 2000-01-01 02:13:20.000381)
With traj AS (
    Select ST_makeTrajectory('STPOINT', 'LINESTRING(0 0, 50 50, 100 100)'::geometry,
                             tsrange('2000-01-01 00:00:00'::timestamp, '2000-01-01 00:01:40'::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]}, "acceleration": {"type": "string", "length": 20, "nullable" : true,"value": ["120","130","140"]}, "active": {"type": "timestamp", "nullable" : false,"value": ["Fri Jan 01 11:35:00 2010", "Fri Jan 01 12:35:00 2010", "Fri Jan 01 13:30:00 2010"]}}, "events": [{"2" : "Fri Jan 02 15:00:00 2010"}, {"3" : "Fri Jan 02 15:30:00 2010"}]}') a
)
SELECT  ST_MakeBox(a), ST_MakeBox(a,'1999-12-31 23:00:00'::timestamp, '2000-01-01 00:00:30'::timestamp) from traj;
                         st_makebox                          |                          st_makebox
-------------------------------------------------------------+-------------------------------------------------------------
 BOX2DT(0 0 2000-01-01 00:00:00,100 100 2000-01-01 00:01:40) | BOX2DT(0 0 2000-01-01 00:00:00,30 30 2000-01-01 00:00:30.000001)
```

5.7.2. ST_MakeBox{Z|T|2D|2DT|3D|3DT}

直接写入值来构建Box。

语法

```
boxndf ST_MakeBoxZ(float8 zmin, float8 zmax);
boxndf ST_MakeBoxT(timestamp tmin, timestamp tmax);
boxndf ST_MakeBox2D(float8 xmin, float8 ymin, float8 xmax, float8 ymax);
boxndf ST_MakeBox2DT(float8 xmin, float8 ymin, timestamp tmin, float8 xmax,float8 yax, time
stamp tmax);
boxndf ST_MakeBox3D(float8 xmin, float8 ymin, float8 zmin, float8 xmax, float8 ymax, float8
zmax);
boxndf ST_MakeBox3DT(float8 xmin, float8 ymin, float8 zmin, timestamp tmin, float8 xmax, fl
oat8 ymax, float8 zmax, timestamp tmax);
```

参数

参数名称	描述
xmin	外包框的x轴下界。
xmax	外包框的x轴上界。
ymin	外包框的y轴下界。
ymax	外包框的y轴上界。
zmin	外包框的z轴下界。
zmax	外包框的z轴上界。
tmin	外包框的时间轴下界。
tmax	外包框的时间轴上界。

描述

根据函数名和指定的参数构建外包框。

由于外包框内部由float类型表示，因此得到的外包框可能会略大于输入的参数，例如下界比实际值略小或上界比实际值略大。

示例

```
SELECT ST_MakeBox2d(0,0,3,3);
  st_makebox2d
----------------
 BOX2D(0 0,3 3)
 SELECT ST_MakeBox3dt(0,0,3,'2000-01-01 00:00:03'::timestamp, 2,5,4,'2000-01-01 02:46:40'::
timestamp);
                              st_makebox3dt
-------------------------------------------------------------------------
 BOX3DT(0 0 3 2000-01-01 00:00:02.999999,2 5 4 2000-01-01 02:46:40.000476)
(1 row)
```

5.7.3. ST_BoxndfToGeom

将外包框类型转化为Geometry类型。

语法

```
geometry ST_BoxNDFToGeom(boxndf box);
```

参数

参数名称	描述
box	指定的外包框。

描述

将外包框类型转化为Geometry类型。

- 如果外包框有z维度，将转化为三维Geometry类型。
- 如果外包框没有z维度，将转化为二维Geometry类型。
- 如果外包框不包含x、y维度，则返回NULL。

示例

```
select ST_AsText(ST_BoxndfToGeom(ST_MakeBox2dt(0,0,'2000-01-01'::timestamp, 20,20, '2020-01
-01'::timestamp)));
             st_astext
-----------------------------------
 POLYGON((0 0,0 20,20 20,20 0,0 0))
```

5.7.4. ST_Has{xy|z|t}

确定指定的外包框是否包含某些维度。

语法

```
bool ST_HasXY(boxndf box);
bool ST_HasZ(boxndf box);
bool ST_HasT(boxndf box);
```

参数

参数名称	描述
box	指定的外包框。

描述

确认指定的外包框是否包含某个或某些维度。由于x维度和y维度为绑定关系，外包框中只会存在全部包含或全不包含的情况。

示例

```
postgres=# select ST_hasz(ST_MakeBox2dt(0,0,'2000-01-01'::timestamp, 20,20, '2020-01-01'::timestamp));
 st_hasz
---------
 f
postgres=# select ST_hast(ST_MakeBox2dt(0,0,'2000-01-01'::timestamp, 20,20, '2020-01-01'::timestamp));
 st_hast
---------
 t
```

5.7.5. ST_{X|Y|Z|T}Min

获取BoxNDF类型中指定维度的最小值。

语法

```
float8 ST_XMin(boxndf box);
float8 ST_YMin(boxndf box);
float8 ST_ZMin(boxndf box);
timestamp ST_TMin(boxndf box);
```

参数

参数名称	描述
box	指定的外包框。

描述

获取外包框类型中指定维度的最小值。如果外包框不包含指定维度时：

- 不包含x维度、y维度和z维度：返回 `NaN` 。
- 不包含t维度：返回 `-infinity` 。

示例

```
select ST_xmin(ST_MakeBox2dt(0,0,'2000-01-01'::timestamp, 20,20, '2020-01-01'::timestamp));
 st_xmin
---------
       0
select ST_zmax(ST_MakeBox2dt(0,0,'2000-01-01'::timestamp, 20,20, '2020-01-01'::timestamp));
 st_zmin
---------
     NaN
select ST_tmax(ST_MakeBox2d(0,0, 20,20));
  st_tmin
-----------
 -infinity
```

5.7.6. ST_{X|Y|Z|T}Max

获取BoxNDF类型中指定维度的最大值。

语法

```
float8 ST_XMax(boxndf box);
float8 ST_YMax(boxndf box);
float8 ST_ZMax(boxndf box);
timestamp ST_TMax(boxndf box);
```

参数

参数名称	描述
box	指定的外包框。

描述

获取外包框类型中指定维度的最大值。如果外包框不包含指定维度时：

- 不包含x维度、y维度和z维度：返回 `NaN` 。
- 不包含t维度：返回 `-infinity` 。

示例

```
select ST_tmax(ST_MakeBox2dt(0,0,'2000-01-01'::timestamp, 20,20, '2020-01-01'::timestamp));
       st_tmax
---------------------
 2020-01-01 00:00:00
select ST_zmin(ST_MakeBox2dt(0,0,'2000-01-01'::timestamp, 20,20, '2020-01-01'::timestamp));
 st_zmin
---------
     NaN
select ST_tmin(ST_MakeBox2d(0,0, 20,20));
  st_tmin
-----------
 -infinity
```

5.7.7. ST_ExpandSpatial

将外包框的空间范围扩大至指定数值。

语法

```
boxndf ST_ExpandSpatial(boxndf box, float8 len);
```

参数

参数名称	描述
box	需要扩大的外包框。
len	需要扩大的长度。

描述

如果外包框中存在xyz三个维度，下界将减少len的长度，上界将增加len的长度。

示例

```
Select ST_ExpandSpatial(ST_MakeBox2dt(0,0,'2000-01-02', 20,20, '2000-03-03'), 4);
                      st_expandspatial
-------------------------------------------------------------
 BOX2DT(-4 -4 2000-01-02 00:00:00,24 24 2000-03-03 00:00:00)
```

5.8. 外包框处理算子

5.8.1. 基本概念

外包框处理算子用来判断算子左侧对象和算子右侧对象两者的时空外包框是否满足指定的时空关系。

算子类别标记

算子类别标记目前分为三种：

- 相交类标记：`&&或&`
- 包含类标记：`@>`
- 被包含类标记：`<@`

> **说明**
> - 当维度标记存在于算子类别标记内部（例如&#&）时，表示算子除了xy二维外还要进行此维度的匹配。
> - 当维度标记存在于算子类别标记两侧（例如#&#）时，表示算子仅考虑此维度，不考虑包含xy轴在内的其它维度。

维度标记

维度标记目前分为两种：

- z维度标记：/
- t维度标记：#

> **说明**
> - 当不存在维度标记时，表示算子为二维。
> - 当两个维度标记同时存在时，z维度标记在前。

5.8.2. 相交类算子

判断左侧对象和右侧对象的外包框是否在指定维度上相交。

语法

```
{geometry, trajectory, boxndf} /&/ {geometry, trajectory, boxndf}
{trajectory, boxndf} #&# {trajectory, boxndf}
{geometry, trajectory, boxndf} && {geometry, trajectory, boxndf}
{geometry, trajectory, boxndf} &/& {geometry, trajectory, boxndf}
{trajectory, boxndf} &#& {trajectory, boxndf}
{trajectory, boxndf} &/#& {trajectory, boxndf}
```

参数

参数名称	描述
算子左侧参数	相交对象。
算子右侧参数	相交对象。

描述

判断左侧对象和右侧对象的外包框（由ST_MakeBox函数生成的BoxNDF类型）是否在指定维度上相交。

支持的相交类算子如下：

- `/&/`：z维度相交。
- `#&#`：t维度相交。
- `&&`：xy二维空间相交。
- `&/&`：xyz三维空间相交。
- `&#&`：xyt二维时空相交。
- `&/#&`：xyzt三维时空相交。

示例

```
WITH box AS(
    SELECT ST_MakeBox3d(0,0,0,5,5,5) a,
           ST_MakeBox3dt(6,6, 3,'2010-04-12 00:00:00',8,8,5,'2013-02-01 00:00:00') b
)
SELECT a /&/ b AS OpZ, a #&# b AS OpT, a && b AS Op2D, a &/& b AS Op3d, a &#& b AS Op2DT, a
&/#& b AS Op3DT from box;
 opz | opt | op2d | op3d | op2dt | op3dt
-----+-----+------+------+-------+-------
 t   | t   | f    | f    | f     | f
```

5.8.3. 包含算子

判断左侧对象的外包框是否在指定维度上包含右侧对象的外包框。

语法

```
{geometry, trajectory, boxndf} /@>/ {geometry, trajectory, boxndf}
{trajectory, boxndf} #@>#  {trajectory, boxndf}
{geometry, trajectory, boxndf} @> {geometry, trajectory, boxndf}
{geometry, trajectory, boxndf} @/> {geometry, trajectory, boxndf}
{trajectory, boxndf} @#> {trajectory, boxndf}
{trajectory, boxndf} @/#> {trajectory, boxndf}
```

参数

参数名称	描述
算子左侧参数	包含对象。
算子右侧参数	被包含对象。

描述

判断左侧对象的外包框（由ST_MakeBox函数生成的BoxNDF类型）是否在指定的维度上包含右侧对象的外包框，与被包含算子含义相反。

支持的包含算子如下：

- `/@>/`：左侧z维度包含右侧z维度。
- `#@>#`：左侧t维度包含右侧t维度。
- `@>`：左侧xy二维空间包含右侧xy二维空间。
- `@&>`：左侧xyz三维空间包含右侧xyz三维空间。
- `@#>`：左侧xyt二维时空包含右侧xyt二维时空。
- `@/#>`：左侧xyzt三维时空包含右侧xyzt三维时空。

示例

```
WITH box AS(
    SELECT ST_MakeBox3dt(0,0,0, '2010-01-01 00:00:00',10,10,10, '2012-01-01 00:00:00') a,
           ST_MakeBox3dt(6,6,3,'2010-01-01 00:00:00',8,8,5,'2013-01-01 00:00:00') b
)
SELECT a /@>/ b AS OpZ, a #@># b AS OpT, a @> b AS Op2D, a @/> b AS Op3D, a @#> b AS Op2DT,
a @/#> b AS Op3DT from box;
 opz | opt | op2d | op3d | op2dt | op3dt
-----+-----+------+------+-------+-------
 t   | f   | t    | t    | f     | f
```

5.8.4. 被包含算子

判断右侧对象的外包框是否在指定维度上包含左侧对象的外包框。

语法

```
{geometry, trajectory, boxndf} /<@/ {geometry, trajectory, boxndf}
{trajectory, boxndf} #<@# {trajectory, boxndf}
{geometry, trajectory, boxndf} <@ {geometry, trajectory, boxndf}
{geometry, trajectory, boxndf} </@ {geometry, trajectory, boxndf}
{trajectory, boxndf} <#@ {trajectory, boxndf}
{trajectory, boxndf} </#@ {trajectory, boxndf}
```

参数

参数名称	描述
算子左侧参数	包含对象。
算子右侧参数	被包含对象。

描述

判断右侧对象的外包框（由ST_MakeBox函数生成的BoxNDF类型）是否在指定的维度上包含左侧对象的外包框，与包含算子含义相反。

支持的被包含算子如下：

- `/<@/`：左侧z维度被右侧z维度包含。
- `#<@#`：左侧t维度被右侧t维度包含。
- `<@`：左侧xy二维空间被右侧xy二维空间包含。
- `<&@`：左侧xyz三维空间被右侧xyz三维空间包含。
- `<#@`：左侧xyt二维时空被右侧xyt二维时空包含。
- `</#@`：左侧xyzt三维时空被右侧xyzt三维时空包含。

示例

```
WITH box AS(
    SELECT ST_MakeBox3dt(0,0,0, '2010-01-01 00:00:00',10,10,10, '2012-01-01 00:00:00') a,
           ST_MakeBox3dt(6,6,3,'2010-01-01 00:00:00',8,8,5,'2013-01-01 00:00:00') b
)
SELECT b /<@/ a AS OpZ, b #<@# a AS OpT, b <@ a AS Op2D, b </@ a AS Op3D, b <#@ a AS Op2DT,
b </#@ a AS Op3DT from box;
 opz | opt | op2d | op3d | op2dt | op3dt
-----+-----+------+------+-------+-------
 t   | f   | t    | t    | f     | f
```

5.9. 空间关系判断

5.9.1. 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', 'LINESTRING(0 0, 5 5, 9 9)'::geometry) from traj_table;
```

5.9.2. 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。
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', 'LINESTRING(0 0, 5 5, 9 9)'::geometry) from traj_table;
```

5.9.3. 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', 'LINESTRING(0 0, 5
5, 9 9)'::geometry, 10) from  traj_table;
```

5.10. 空间处理

5.10.1. 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。

参数名称	描述
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', 'LINESTRING(0 0, 5 5
, 9 9)'::geometry) from  traj_table;
```

5.10.2. 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。
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', 'LINESTRING(0 0, 5 5,
9 9)'::geometry) from  traj_table;
```

5.11. 空间统计

5.11.1. 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', 'LINESTRING(
0 0, 5 5, 9 9)'::geometry) from  traj_table;
```

5.11.2. 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。

参数名称	描述
g	几何对象。

示例

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

5.12. 时空关系判断

5.12.1. 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');
```

5.12.2. 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 whe
re id=2), '2010-1-1 13:00:00', '2010-1-1 14:00:00');
```

5.12.3. 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, f
loat8 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。
d	指定距离。

示例

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

5.12.4. 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, i
nterval 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_t
able where id=2), '2010-1-1 13:00:00', '2010-1-1 14:00:00', INTERVAL '30s');
```

5.12.5. ST_{Z|T|2D|2DT|3D|3DT}Intersects

ST_ndIntersects函数，判断指定的两个对象在指定的坐标轴上是否相交。

语法

```
bool ST_{2D|3D}Intersects(geometry geom, trajectory traj);
bool ST_{2D|3D}Intersects(trajectory traj, geometry geom);
bool ST_{2D|3D}Intersects(geometry geom, trajectory traj, timestamp ts, timestamp te);
bool ST_{2D|3D}Intersects(trajectory traj, geometry geom, timestamp ts, timestamp te);
bool ST_{Z|T|2D|2DT|3D|3DT}Intersects(boxndf box, trajectory traj);
bool ST_{Z|T|2D|2DT|3D|3DT}Intersects(trajectory traj, boxndf box);
bool ST_{Z|T|2D|2DT|3D|3DT}Intersects(boxndf box, trajectory traj, timestamp ts, timestamp
te);
bool ST_{Z|T|2D|2DT|3D|3DT}Intersects(trajectory traj, boxndf box, timestamp ts, timestamp
te);
bool ST_{2D|2DT|3D|3DT}Intersects(trajectory traj1, trajectory traj2);
bool ST_{2D|2DT|3D|3DT}Intersects(trajectory traj1, trajectory traj2, timestamp ts, timesta
mp te);
```

参数

参数名称	描述
geom	需要判断的几何对象。
traj	需要判断的轨迹对象，或者产生子轨迹的原始轨迹。
traj1	需要判断的轨迹对象，或者产生子轨迹的原始轨迹。
traj2	需要判断的轨迹对象，或者产生子轨迹的原始轨迹。
box	需要判断的外包框对象。
ts	如存在，表示按此时间作为开始时间截取子轨迹。
te	如存在，表示按此时间作为结束时间截取子轨迹。

描述

在指定维度上，返回需要判断的两个对象（即第一个和第二个参数）的相交结果。对于没有值的维度，将其视为任意值进行判断。

如果函数包含ts和te参数，表示需要判断的对象（traj，或traj1和traj2）是在[ts,te]时间段内的子轨迹；如果函数不包含ts和te参数，表示需要判断的对象（traj，或traj1和traj2）是完整的轨迹。

直接调用 `ST_Intersects(...)` 函数相当于调用 `ST_2DIntersects(...)` （若前两个参数其中一个为几何对象）或 `ST_3DTIntersects(...)` （若前两个参数均为轨迹对象）。

示例

```
WITH traj AS(
    SELECT (' {"trajectory":{"version":1,"type":"STPOINT","leafcount":6,"start_time":"2000-
01-01 03:15:42","end_time":"2000-01-01 05:16:43",' ||
            '"spatial":"LINESTRING(2 2 0,33.042158099636 36.832684322819 0,47.244002354518
47.230026333034 0,64.978971942887 60.618813472986 0,77.621717839502 78.012496630661 0,80 78
0)",' ||
            '"timeline":["2000-01-01 03:15:42","2000-01-01 03:39:54","2000-01-01 04:04:06",
"2000-01-01 04:28:18","2000-01-01 04:52:31","2000-01-01 05:16:43"]}}')::trajectory a,
           ('{"trajectory":{"version":1,"type":"STPOINT","leafcount":4,"start_time":"2000-0
1-01 02:17:58.656079","end_time":"2000-01-01 03:43:59.620923",' ||
            '"spatial":"LINESTRING(40 2 0,15.17549143297 51.766017656152 0,1.444002354518 6
9.630026333034 0,3 70 0)","timeline":["2000-01-01 02:17:58.656079",' ||
            '"2000-01-01 02:46:38.977693","2000-01-01 03:15:19.299307","2000-01-01 03:43:59
.620923"]}}')::trajectory b
)
SELECT ST_2dIntersects(a,b), ST_2dtIntersects(a,b), ST_3dIntersects(a,b), ST_3dtIntersects(
a,b) from traj;
 st_2dintersects | st_2dtintersects | st_3dintersects | st_3dtintersects
-----------------+------------------+-----------------+------------------
 t               | f                | t               | f
```

5.12.6. ST_{2D|2DT|3D|3DT}Intersects_IndexLeft

ST_ndIntersects_IndexLeft函数，判断指定的两个对象在指定的坐标轴上是否相交，并指定使用第一个参数所在的列的索引。

语法

```
bool ST_{2D|2DT|3D|3DT}Intersects_IndexLeft(trajectory traj1, trajectory traj2, timestamp ts, timestamp te);
```

参数

参数名称	描述
traj1	需要判断的轨迹对象，或者产生子轨迹的原始轨迹。
traj2	需要判断的轨迹对象，或者产生子轨迹的原始轨迹。
ts	如存在，表示按此时间作为开始时间截取子轨迹。
te	如存在，表示按此时间作为结束时间截取子轨迹。

描述

当ST_ndIntersects函数需要判断两条子轨迹是否相交时（即调用 `ST_{2D|2DT|3D|3DT}Intersects(trajectory traj1, trajectory traj2, timestamp ts, timestamp te)` 函数时），函数无法自主判断要使用两个轨迹列哪一列的索引，因此会带来额外的计算开销。

我们可以使用 `ST_ndIntersects_IndexLeft(...)` 函数，手工指定使用第一个参数所在的列上的索引，降低计算开销。

示例

```
EXPLAIN SELECT count(*) FROM sqltr_test_trajs WHERE ST_2DIntersects_IndexLeft(traj, ST_make
Trajectory('STPOINT', 'LINESTRING(-70 -30, -72 -34, -73 -35)'::geometry, '[2000-01-01 00:01
:30, 2000-01-01 00:15:00)'::tsrange,
  '{"leafcount":3,"attributes":{"velocity": {"type": "integer", "length": 2,"nullable" : tr
ue,"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,"valu
e": ["120","130","140"]}, "active": {"type": "timestamp", "nullable" : false,"value": ["Fri
Jan 01 11:35:00 2010", "Fri Jan 01 12:35:00 2010", "Fri Jan 01 13:30:00 2010"]}}, "events":
[{"2" : "Fri Jan 02 15:00:00 2010"}, {"3" : "Fri Jan 02 15:30:00 2010"}]}'), ST_PGEpochToTS
(0), ST_PGEpochToTs(7000));
Aggregate  (cost=4201.04..4201.05 rows=1 width=8)
   ->  Bitmap Heap Scan on sqltr_test_trajs  (cost=98.64..4199.77 rows=505 width=0)
         Recheck Cond: ('BOX2DT(-73 -35 2000-01-01 00:01:29.999994,-70 -30 2000-01-01 00:15
:00)'::boxndf &#& traj)
         Filter: _st_2dintersects(traj, '{"trajectory":{"version":1,"type":"STPOINT","leafc
ount":3,"start_time":"2000-01-01 00:01:30","end_time":"2000-01-01 00:15:00","spatial":"LINE
STRING(-70 -30,-72 -34,-73 -35)","timeline":["2000-01-01 00:01:30","2000-01-01 00:08:15","2
000-01-01 00:15:00"],"attributes":{"leafcount":3,"velocity":{"type":"integer","length":2,"n
ullable":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]},"acceleration":{"type":"string","length":20,"nullable":true,"value":[
"120","130","140"]},"active":{"type":"timestamp","length":8,"nullable":false,"value":["2010
-01-01 11:35:00","2010-01-01 12:35:00","2010-01-01 13:30:00"]}},"events":[{"2":"2010-01-02
15:00:00"},{"3":"2010-01-02 15:30:00"}]}}'::trajectory, '2000-01-01 00:00:00'::timestamp wi
thout time zone, '2000-01-01 01:56:40'::timestamp without time zone)
         ->  Bitmap Index Scan on sqltr_test_traj_gist_2dt  (cost=0.00..98.51 rows=1515 wid
th=0)
               Index Cond: (traj &#& 'BOX2DT(-73 -35 2000-01-01 00:01:29.999994,-70 -30 200
0-01-01 00:15:00)'::boxndf)
```

5.12.7. ST_{2D|2DT|3D|3DT}DWithin

ST_ndDWithin函数，判断指定的两个对象在指定的坐标轴上是否距离小于等于某一给定值。

语法

```
bool ST_{2D|3D}DWithin(geometry geom, trajectory traj, float8 dist);
bool ST_{2D|3D}DWithin(trajectory traj, geometry geom, float8 dist);
bool ST_{2D|3D}DWithin(geometry geom, trajectory traj, timestamp ts, timestamp te, float8 d
ist);
bool ST_{2D|3D}DWithin(trajectory traj, geometry geom, timestamp ts, timestamp te, float8 d
ist);
bool ST_{2D|2DT|3D|3DT}DWithin(boxndf box, trajectory traj, float8 dist);
bool ST_{2D|2DT|3D|3DT}DWithin(trajectory traj, boxndf box, float8 dist);
bool ST_{2D|2DT|3D|3DT}DWithin(boxndf box, trajectory traj, timestamp ts, timestamp te, flo
at8 dist);
bool ST_{2D|2DT|3D|3DT}DWithin(trajectory traj, boxndf box, timestamp ts, timestamp te, flo
at8 dist);
bool ST_{2D|2DT|3D|3DT}DWithin(trajectory traj1, trajectory traj2, float8 dist);
bool ST_{2D|2DT|3D|3DT}DWithin(trajectory traj1, trajectory traj2, timestamp ts, timestamp
te, float8 dist);
```

参数

参数名称	描述
geom	需要判断的几何对象。
traj	需要判断的轨迹对象，或者产生子轨迹的原始轨迹。
traj1	需要判断的轨迹对象，或者产生子轨迹的原始轨迹。
traj2	需要判断的轨迹对象，或者产生子轨迹的原始轨迹。
box	需要判断的外包框对象。
ts	如存在，表示按此时间作为开始时间截取子轨迹。
te	如存在，表示按此时间作为结束时间截取子轨迹。
dist	距离的临界值。

描述

- 对于2D和3D的ST_ndDWithin操作，我们判断给定的两个对象（即第一个和第二个参数）在二维/三维空间上的投影（即对应的geometry类型）是否距离小于等于给定值。
- 对于2DT和3DT的ST_ndDWithin操作，我们判断两者是否在某一时间点上在指定维度上的距离小于等于过给定值。

如果函数包含ts和te参数，代表判断ST_ndDWithin的轨迹对象（traj，或traj1和traj2）是在[ts,te]时间段内的子轨迹；否则则表示需要判断的轨迹对象（traj，或traj1和traj2）是完整的轨迹。

直接调用 `ST_DistanceWithin(..)` 函数相当于调用 `ST_2DDWithin(...)` （若前两个参数其中一个为几何对象）或 `ST_3DTDWithin(...)` （若前两个参数均为轨迹对象）。

示例

```
WITH traj AS(
    Select ST_makeTrajectory('STPOINT', 'LINESTRING(0 0 10, 50 0 10, 100 0 10)'::geometry,
                             ('[' || ST_PGEpochToTS(0) || ',' || ST_PGEpochToTS(100) || ')'
)::tsrange,
                             '{"leafcount":3,"attributes":{"velocity": {"type": "integer",
"length": 2,"nullable" : true,"value": [120,130,140]}, "accuracy": {"type": "float", "lengt
h": 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", "nulla
ble" : false,"value": ["Fri Jan 01 11:35:00 2010", "Fri Jan 01 12:35:00 2010", "Fri Jan 01
13:30:00 2010"]}}, "events": [{"2" : "Fri Jan 02 15:00:00 2010"}, {"3" : "Fri Jan 02 15:30:
00 2010"}]}') a,
           ST_makeTrajectory('STPOINT', 'LINESTRING(0 20, 50 20, 100 20)'::geometry,
                             ('[' || ST_PGEpochToTS(0) || ',' || ST_PGEpochToTS(100) ||')')
::tsrange,
                             '{"leafcount":3,"attributes":{"velocity": {"type": "integer",
"length": 2,"nullable" : true,"value": [120,130,140]}, "accuracy": {"type": "float", "lengt
h": 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", "nulla
ble" : false,"value": ["Fri Jan 01 11:35:00 2010", "Fri Jan 01 12:35:00 2010", "Fri Jan 01
13:30:00 2010"]}}, "events": [{"2" : "Fri Jan 02 15:00:00 2010"}, {"3" : "Fri Jan 02 15:30:
00 2010"}]}') b
)
SELECT ST_2dDWithin(a,b, ST_PGEpochToTS(0),ST_PGEpochToTS(49), 19), ST_3dDWithin(a,b, ST_PG
EpochToTS(0),ST_PGEpochToTS(49), 19),
       ST_2dDWithin(a,b, ST_PGEpochToTS(0),ST_PGEpochToTS(50), 20), ST_3dDWithin(a,b, ST_PG
EpochToTS(0),ST_PGEpochToTS(50), 20),
       ST_2dDWithin(a,b, ST_PGEpochToTS(0),ST_PGEpochToTS(70), 21), ST_3dDWithin(a,b, ST_PG
EpochToTS(0),ST_PGEpochToTS(70), 21),
       ST_2dtDWithin(a,b, ST_PGEpochToTS(0),ST_PGEpochToTS(49), 19), ST_3dtDWithin(a,b, ST_
PGEpochToTS(0),ST_PGEpochToTS(49), 19),
       ST_2dtDWithin(a,b, ST_PGEpochToTS(0),ST_PGEpochToTS(50), 20), ST_3dtDWithin(a,b, ST_
PGEpochToTS(0),ST_PGEpochToTS(50), 20),
       ST_2dtDWithin(a,b, ST_PGEpochToTS(0),ST_PGEpochToTS(70), 21), ST_3dtDWithin(a,b, ST_
PGEpochToTS(0),ST_PGEpochToTS(70), 21) from traj;
NOTICE:  One or both of the geometries is missing z-value. The unknown z-value will be rega
rded as "any value"
NOTICE:  One or both of the geometries is missing z-value. The unknown z-value will be rega
rded as "any value"
 st_2ddwithin | st_3ddwithin | st_2ddwithin | st_3ddwithin | st_2ddwithin | st_3ddwithin |
st_2dtdwithin | st_3dtdwithin | st_2dtdwithin | st_3dtdwithin | st_2dtdwithin | st_3dtdwith
in
--------------+--------------+--------------+--------------+--------------+--------------+-
--------------+---------------+---------------+---------------+---------------+------------
---
 f            | f            | t            | t            | t            | t            |
f             | f             | t             | t             | t             | t
```

5.12.8. ST_{2D|2DT|3D|3DT}DWithin_IndexLeft

ST_ndDwithin_IndexLeft函数，判断指定的两个对象在指定的坐标轴上是否距离小于等于某一给定值，并指定使用第一个参数所在的列的索引。

语法

```
bool ST_{2D|2DT|3D|3DT}DWithin_IndexLeft(trajectory traj1, trajectory traj2, float8 dist);
bool ST_{2D|2DT|3D|3DT}DWithin_IndexLeft(trajectory traj1, trajectory traj2, timestamp ts,
timestamp te, float8 dist);
```

参数

参数名称	描述
traj1	需要判断的轨迹对象，或者产生子轨迹的原始轨迹。
traj2	需要判断的轨迹对象，或者产生子轨迹的原始轨迹。
ts	如存在，表示按此时间作为开始时间截取子轨迹。
te	如存在，表示按此时间作为结束时间截取子轨迹。
dist	距离的临界值。

描述

当 `ST_ndDWithin(...)` 操作涉及两个轨迹列时（即调用 `ST_{2D|2DT|3D|3DT}DWithin(trajectory traj1, trajectory traj2, float8 dist)` 或 `ST_{2D|2DT|3D|3DT}DWithin(trajectory traj1, trajectory traj2, timestamp ts, timestamp te, float8 dist)` 函数时），数据库无法判断应当使用哪个列的索引，因此会带来额外的计算开销。

我们可以使用ST_ndDwithin_IndexLeft函数手工指定使用第一个参数所对应的列上的索引，以降低计算开销。

示例

```
EXPLAIN SELECT count(*) FROM sqltr_test_trajs WHERE ST_3DTDWithin_IndexLeft(traj, ST_makeTrajectory('STPOINT', 'LINESTRING(-70 -30, -72 -34, -73 -35)'::geometry, '[2000-01-01 00:01:30, 2000-01-01 00:15:00)'::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 11:35:00 2010", "Fri Jan 01 12:35:00 2010", "Fri Jan 01 13:30:00 2010"]}}, "events": [{"2" : "Fri Jan 02 15:00:00 2010"}, {"3" : "Fri Jan 02 15:30:00 2010"}]}'), 2);
Aggregate  (cost=4341.89..4341.90 rows=1 width=8)
   ->  Bitmap Heap Scan on sqltr_test_trajs  (cost=103.31..4340.56 rows=529 width=0)
         Recheck Cond: ('BOX2DT(-75 -37 2000-01-01 00:01:29.999994,-68 -28 2000-01-01 00:15:00)'::boxndf &/#& traj)
         Filter: _st_3dtdwithin(traj, '{"trajectory":{"version":1,"type":"STPOINT","leafcount":3,"start_time":"2000-01-01 00:01:30","end_time":"2000-01-01 00:15:00","spatial":"LINESTRING(-70 -30,-72 -34,-73 -35)","timeline":["2000-01-01 00:01:30","2000-01-01 00:08:15","2000-01-01 00:15:00"],"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]},"acceleration":{"type":"string","length":20,"nullable":true,"value":["120","130","140"]},"active":{"type":"timestamp","length":8,"nullable":false,"value":["2010-01-01 11:35:00","2010-01-01 12:35:00","2010-01-01 13:30:00"]}},"events":[{"2":"2010-01-02 15:00:00"},{"3":"2010-01-02 15:30:00"}]}}'::trajectory, '2'::double precision)
         ->  Bitmap Index Scan on sqltr_test_traj_gist_2dt  (cost=0.00..103.18 rows=1586 width=0)
               Index Cond: (traj &/#& 'BOX2DT(-75 -37 2000-01-01 00:01:29.999994,-68 -28 2000-01-01 00:15:00)'::boxndf)
```

5.12.9. ST_{T|2D|2DT|3D|3DT}Contains

ST_ndContains函数，判断第一个参数所对应的对象在指定的坐标轴上是否包含第二个参数所对应的对象。

语法

```
bool ST_TContains(tsrange r, trajectory traj);
bool ST_TContains(trajectory traj, tsrange r);
bool ST_2DContains(geometry geom, trajectory traj);
bool ST_2DContains(trajectory traj, geometry geom);
bool ST_2DContains(geometry geom, trajectory traj, timestamp ts, timestamp te);
bool ST_2DContains(trajectory traj, geometry geom, timestamp ts, timestamp te);
bool ST_{2D|2DT|3D|3DT}Contains(boxndf box, trajectory traj);
bool ST_{2D|2DT|3D|3DT}Contains(boxndf box, trajectory traj, timestamp ts, timestamp te);
```

参数

参数名称	描述
geom	需要判断的几何对象。
traj	需要判断的轨迹对象，或者产生子轨迹的原始轨迹。

参数名称	描述
box	需要判断的外包框对象。
r	需要判读的时间范围。
ts	如存在，表示按此时间作为开始时间截取子轨迹。
te	如存在，表示按此时间作为结束时间截取子轨迹。

描述

判断第一个参数是否包含第二个参数。

- 对于geometry类型，目前仅支持二维的操作，即判断轨迹或一定时间段内的子轨迹的二维投影，是否包含或被包含于给定的geom内。
- 对于boxndf作为第一个参数，trajectory作为第二个参数的Contains函数，将判断各个维度轨迹（或子轨迹）在指定的维度上是否在给定的外包框范围内。如果外包框、轨迹或子轨迹不包含某个给定的维度，则此维度将取任意值，即对此坐标轴自动满足contains条件。

> **说明** 部分geometry类型（如POLYHEDRALSURFACE）目前不支持ST_Contains操作。

示例

```
WITH traj AS(
    Select ST_makeTrajectory('STPOINT', 'LINESTRING(0 0, 50 50, 100 100)'::geometry,
                             ('[' || ST_PGEpochToTS(0) || ',' || ST_PGEpochToTS(100) || ')'
)::tsrange,
                             '{"leafcount":3,"attributes":{"velocity": {"type": "integer",
"length": 2,"nullable" : true,"value": [120,130,140]}, "accuracy": {"type": "float", "lengt
h": 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", "nulla
ble" : false,"value": ["Fri Jan 01 11:35:00 2010", "Fri Jan 01 12:35:00 2010", "Fri Jan 01
13:30:00 2010"]}}, "events": [{"2" : "Fri Jan 02 15:00:00 2010"}, {"3" : "Fri Jan 02 15:30:
00 2010"}]}') b,
           ST_MakeBox3dt(0,0,0,ST_PgEpochToTS(0), 50,50,50, ST_PgEpochToTS(49)) a
)
SELECT ST_2dContains(a,b,ST_PGEpochToTS(0), ST_PGEpochToTS(49)), ST_3dContains(a,b,ST_PGEpo
chToTS(0), ST_PGEpochToTS(49)),
       ST_2dtContains(a,b,ST_PGEpochToTS(0), ST_PGEpochToTS(49)), ST_3dtContains(a,b,ST_PGE
pochToTS(0), ST_PGEpochToTS(49)),
       ST_2dContains(a,b,ST_PGEpochToTS(0), ST_PGEpochToTS(50)), ST_3dContains(a,b,ST_PGEpo
chToTS(0), ST_PGEpochToTS(50)),
       ST_2dtContains(a,b,ST_PGEpochToTS(0), ST_PGEpochToTS(50)), ST_3dtContains(a,b,ST_PGE
pochToTS(0), ST_PGEpochToTS(50)),
       ST_2dContains(a,b,ST_PGEpochToTS(0), ST_PGEpochToTS(70)), ST_3dContains(a,b,ST_PGEpo
chToTS(0), ST_PGEpochToTS(70)),
       ST_2dtContains(a,b,ST_PGEpochToTS(0), ST_PGEpochToTS(70)), ST_3dtContains(a,b,ST_PGE
pochToTS(0), ST_PGEpochToTS(70)) from traj;
 st_2dcontains | st_3dcontains | st_2dtcontains | st_3dtcontains | st_2dcontains | st_3dcon
tains | st_2dtcontains | st_3dtcontains | st_2dcontains | st_3dcontains | st_2dtcontains |
st_3dtcontains
---------------+---------------+----------------+----------------+---------------+---------
------+----------------+---------------+---------------+---------------+----------------+-
---------------
 t             | t             | t              | t              | t             | t
| f             | f             | f              | f              | f             | f
```

5.12.10. ST_{T|2D|2DT|3D|3DT}Within

ST_ndWithin函数，判断第一个参数所对应的对象在指定的坐标轴上是否被包含于第二个参数所对应的对象。

语法

```
bool ST_TWithin(tsrange r, trajectory traj);
bool ST_TWithin(trajectory traj, tsrange r);
bool ST_2DWithin(geometry geom, trajectory traj);
bool ST_2DWithin(trajectory traj, geometry geom);
bool ST_2DWithin(geometry geom, trajectory traj, timestamp ts, timestamp te);
bool ST_2DWithin(trajectory traj, geometry geom, timestamp ts, timestamp te);
bool ST_{2D|2DT|3D|3DT}Within(trajectory traj, boxndf box);
bool ST_{2D|2DT|3D|3DT}Within(trajectory traj, boxndf box, timestamp ts, timestamp te);
```

参数

参数名称	描述
geom	需要判断的几何对象。
traj	需要判断的轨迹对象，或者产生子轨迹的原始轨迹。
box	需要判断的外包框对象。
r	需要判断的时间范围。
ts	如存在，表示按此时间作为开始时间截取子轨迹。
te	如存在，表示按此时间作为结束时间截取子轨迹。

描述

判断第一个参数是否被包含于第二个参数，等价于将第一个参数与第二个参数交换的ST_ndContains函数。

> **说明** 部分geometry类型（如POLYHEDRALSURFACE）目前不支持ST_Within操作。

示例

```
WITH traj AS(
    SELECT (' {"trajectory":{"version":1,"type":"STPOINT","leafcount":6,"start_time":"2000-
01-01 03:15:42","end_time":"2000-01-01 05:16:43",' ||
            '"spatial":"LINESTRING(2 2 0,33.042158099636 36.832684322819 0,47.244002354518
47.230026333034 0,64.978971942887 60.618813472986 0,77.621717839502 78.012496630661 0,80 78
0)",' ||
            '"timeline":["2000-01-01 03:15:42","2000-01-01 03:39:54","2000-01-01 04:04:06",
"2000-01-01 04:28:18","2000-01-01 04:52:31","2000-01-01 05:16:43"]}}')::trajectory a,
          'LINESTRING(2 2 0,33.042158099636 36.832684322819 0,47.244002354518 47.230026333
034 0,64.978971942887 60.618813472986 0,77.621717839502 78.012496630661 0,80 78 0)'::geomet
ry b
)
SELECT ST_2dWithin(b,a) from traj;
 st_2dwithin
-------------
 t
```

5.13. 时空处理

5.13.1. 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');
```

5.14. 时空统计

5.14.1. 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。
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');
```

5.14.2. 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, timesta
mp 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 fr
om traj_table where id=2), '2010-1-1 13:00:00', '2010-1-1 14:00:00');
```

5.15. 距离测量

5.15.1. 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)
```

5.15.2. 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 tra
j_table where id=2));
```

5.15.3. 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));
```

5.16. 相似度分析

5.16.1. ST_lcsSimilarity

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

语法

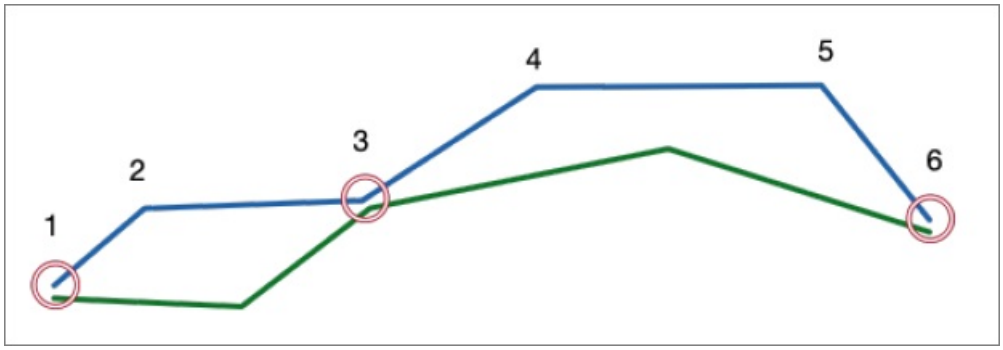
```
integer ST_lcsSimilarity(trajectory traj1, trajectory traj2, float8 dist, distanceUnit unit default 'M' );
integer ST_lcsSimilarity(trajectory traj1, trajectory traj2, float8 dist, interval lag, distanceUnit unit default 'M');
```

参数

参数名称	描述
traj1	轨迹对象1。
traj2	轨迹对象2。
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 , 114.00053
5 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 , 114.00053
5 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 , 114.00053
5 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 , 114.00053
5 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'], NUL
L) b)
Select st_LCSSimilarity(a, b, 100, interval '30 seconds') from traj;
 st_lcssimilarity
-------------------
2
(1 row)
```

5.16.2. 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 d
efault 'M');
float8 ST_lcsDisatance(trajectory traj1, trajectory traj2, float8 dist, interval lag, dista
nceUnit unit default 'M');
```

参数

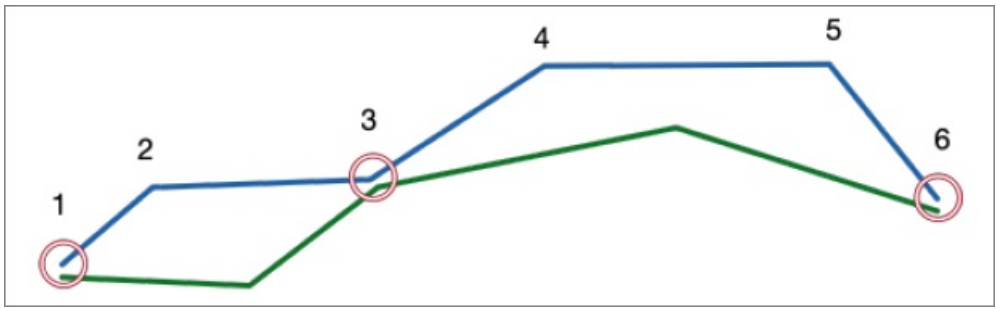
参数名称	描述
traj1	轨迹对象1。

参数名称	描述
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 , 114.00053
5 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 , 114.00053
5 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 , 114.00053
5 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 , 114.00053
5 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'], NUL
L) 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 , 114.00053
5 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 , 114.00053
5 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'], NUL
L) b)
Select st_LCSDistance(a, b, 100, interval '30 seconds', 'M') from traj;
 st_lcsdistance
-------------------
0.666666666666667
(1 row)
```

5.16.3. ST_lcsSubDistance

计算LCSS轨迹段与traj1在这段轨迹上的距离。

语法

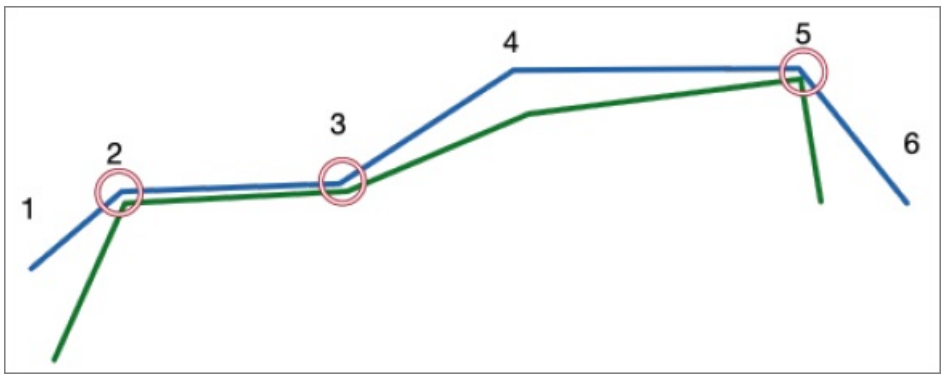
```
float8 ST_lcsSubDisatance(trajectory traj1, trajectory traj2, float8 dist, distanceUnit unit default 'M');
float8 ST_lcsSubDisatance(trajectory traj1, trajectory traj2, float8 dist, interval lag, distanceUnit unit default 'M');
```

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 , 114.00053
5 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 , 114.00053
5 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 , 114.00053
5 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 , 114.00053
5 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)
```

5.17. 索引方式

5.17.1. GisT索引

对轨迹数据列创建GisT索引。

语法

```
CREATE INDEX [index_name] on table_name USING GIST(traj_col [operator_family]);
```

- index_name：索引名，可以省略。
- table_name：表名。
- traj_col：轨迹列名。
- operator_family：指定索引所使用的算子族，可以省略，默认值为trajgist_ops_3dt。

> **说明** 建立索引后，可以加速各类算子以及ST_ndIntersect、ST_ndDWithin、ST_ndContains、ST_ndWithin函数的查询。

支持算子族

当前支持6个索引的算子族，说明如下。

算子族名称	描述
trajgist_ops_z	对z轴范围建立索引，支持仅包含z轴的查询。
trajgist_ops_t	对t轴范围建立索引，支持仅包含t轴的查询。
trajgist_ops_2d	对x、y轴范围建立索引，支持仅包含x、y轴信息的查询。
trajgist_ops_2dt	对x、y、t轴范围建立索引，支持二维、时间，以及混合查询。
trajgist_ops_3d	对x、y、z轴范围建立索引，支持二维、三维、z轴查询。
trajgist_ops_3dt	对x、y、z、t轴范围建立索引，支持以上全部查询。

> **说明** 对于同一个查询，一般来说索引所包含的维度越少，最终的查询效率越高。因此当频繁使用某类查询时，建议在默认的trajgist_ops_3dt之外建立对应维度的查询。

示例

```
--建立时间维索引。
CREATE INDEX on table_name USING GIST(traj_col trajgist_ops_t);
--建立二维索引。
CREATE INDEX on table_name USING GIST(traj_col trajgist_ops_2d);
--建立二维时空复合索引。
CREATE INDEX on table_name USING GIST(traj_col trajgist_ops_2dt);
```

5.17.2. TrajGisT索引

TrajGisT索引是GisT索引的扩展。

背景信息

在GisT基础上，TrajGisT提供额外两点优化：

- TrajGisT对索引的开销估计进行了优化，当建立了多个索引时，TrajGisT可以比GisT更好地在不同索引之间进行选择。
- TrajGist支持索引的向上兼容，即当索引所记录的信息不足以对查询进行精确判断时，仍然可以通过索引扫描得到一个粗粒度的结果。例如只建立了时间索引，但需要进行2维和时间的2DTIntersects操作时，可以用时间索引过滤掉和给定的时间段不相交的轨迹。

语法

```
CREATE INDEX [index_name] on table_name USING TRAJGIST(traj_col [operator_family]);
```

- index_name：索引名，可以省略。
- table_name：表名。
- traj_col：轨迹列名。
- operator_family：指定索引所使用的算子族，可以省略，默认值为trajgist_ops_3dt。

> **说明** 建立索引后，可以加速各类算子以及ST_ndIntersect、ST_ndDWithin、ST_ndContains、ST_ndWithin函数的查询。

支持算子族

当前支持6个索引的算子族，说明如下。

算子族名称	描述
trajgist_ops_z	对z轴范围建立索引，支持所有包含z轴的查询。
trajgist_ops_t	对t轴范围建立索引，支持所有包含t轴的查询。
trajgist_ops_2d	对x、y轴范围建立索引，支持所有包含x、y轴信息的查询。
trajgist_ops_2dt	对x、y、t轴范围建立索引，支持所有包含x、y、t轴的查询。
trajgist_ops_3d	对x、y、z轴范围建立索引，支持所有包含x、y、z轴的查询。
trajgist_ops_3dt	对x、y、z、t轴范围建立索引，支持以上全部查询。

> **说明** TrajGist目前仅支持单列索引，不能与其它（非轨迹）列建立多列索引。

5.18. 变量

5.18.1. 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;
```

6.常见问题

6.1. 加载栅格数据

本文档介绍如何使用ST_ImportFrom函数将本地文件系统或OSS上的栅格影像文件加载到Ganos中。

注意事项

实例必须可以访问到栅格影像，如果是OSS数据，请确保云数据库与OSS数据所在地域相同。相关信息请参见OSS访问域名使用规则信息。

ST_ImportFrom函数

ST_ImportFrom是一个Ganos Raster函数，可以将一个存放在本地文件系统或OSS上的栅格影像文件加载到Ganos中。

> **说明**
> - 以下仅为最基本的加载语法和说明，更多功能请参见工具对应文档。
> - 关于ST_ImportFrom具体参数解释，详情请参见ST_ImportFrom。

语法

```
INSERT INTO <raster_table_name>(<raster_column_name>) VALUES (ST_IMPORTFROM('<chunk_table_name>','<path_to_raster_file>'))
```

示例

1. 创建ganos_raster扩展。

   ```
   CREATE EXTENSION Ganos_Raster CASCADE
   ```

2. 将一个本地文件上的栅格文件加载到Ganos上 。

   ```
   INSERT INTO raster_table(name, rast) VALUES ('local_file', ST_ImportFrom('chunk_table', '/home/beijing.tif'));
   ```

3. 将一个OSS文件上的栅格影像文件加载到Ganos上。

   ```
   INSERT INTO raster_table(name, rast) VALUES ('oss_file', ST_ImportFrom('chunk_table', 'oss://ak:as@bucket/data/beijing.tif'));
   ```

6.2. 加载矢量数据

本文介绍如何将矢量数据加载到Ganos中，建议您使用的工具为shp2pgsql、ogr2ogr或QGIS。

准备工作

在加载矢量数据之前，请确保在数据库中已输入如下命令，来创建ganos_geometry扩展：

```
CREATE EXTENSION ganos_geometry CASCADE
```

shp2pgsql命令行工具

shp2pgsql命令行工具是一个将Esri Shapefile文件转换为SQL文件的命令行工具。通过将shapfile转换为SQL文件，可以实现将Shapefile数据加载到Ganos中。

- 语法

```
shp2pgsql -s <srid> -c -W <charset> <path_to_shpfile> <schema>.<table_name> | psql -d <db name> -h <host> -U <user_name> -p <port>
```

- 参数说明

参数名称	描述	示例
-s <srid>	空间参考ID。	4490
-c	创建一张新表。 您也可以选择其他模式： ○ -a：追加 ○ -d：加载数据之前先删除表 ○ -p：仅创建表结构SQL，不写入空间数据	无
-W <charset>	shapefile字符集。	"GBK" 或 "UTF8"
<path_to_shpfile>	shapefile 文件路径，需要被shp2pgsql命令工具访问到。	/home/roads.shp
<schema>.<table_name>	创建的geometry表的名称。	public.roads
-d <dbname>	数据库名称。	mydb
-h <host>	数据库实例地址。	pgm-xxxxxxx.pg.rds.aliyuncs.com
-U <user_name>	用户名。	my_name
-p <port>	端口号。	3433

- 示例

```
shp2pgsql -s 4490 -c -W "GBK" /home/roads.shp public.roads | psql -d mydb -h pgm-xxxxxxx.pg.rds.aliyuncs.com -U my_name -p 3433
```

> **说明** 如果需要在脚本中批量导入，为防止频繁提示输入密码，可以在脚本中设置环境变量PGPASSWORD：`export PGPASSWORD=my_pass`。

ogr2ogr命令行工具

ogr2ogr是GDAL/OGR提供的矢量数据转换工具，支持Esri Shapefile，MapInfo和FileGDB等常见矢量数据类型。矢量数据类型参见 矢量数据类型。

> **说明** 使用ogr2ogr命令行工具加载矢量数据，请先确保GDAL支持postgis驱动格式。

- 语法

```
ogr2ogr -nln <table_name> -f PostgreSQL PG:"dbname='<dbname>' host='<host>' port='<port>' user='<user_name>' password='<password>'" -lco SPATIAL_INDEX=GIST -lco FID=<FID_NAME> -overwrite "<PATH_TO_FILE>" -nlt GEOMETRY
```

- 参数说明

参数名称	描述	示例
-nln <table_name>	矢量表名称。	roads
-f PostgreSQL PG:"dbname='<dbname>' host='<host>' port='<port>' user='<user_name>' password='<password>'"	PostgreSQL连接信息。	dbname='mydb' host='pgm-xxxxxxx.pg.rds.aliyuncs.com' port='3433' user='my_name' password='my_pass'
-lco SPATIAL_INDEX=GIST	指定需要创建GisT空间索引。	无
-lco FID=<FID_NAME>	指定fid名称。	fid
-overwrite	改写模式。 您也可以选择其他模式： -append追加模式。	无
<PATH_TO_FILE>	矢量数据文件路径，请确保ogr2ogr命令行工具拥有访问权限。	/home/roads.shp /home/land.tab
-nlt GEOMETRY	指定几何类型为geometry，在加载多边形时最好指定，防止出现multipolygon类型与非multipolygon在同一个文件中加载出错问题。	无

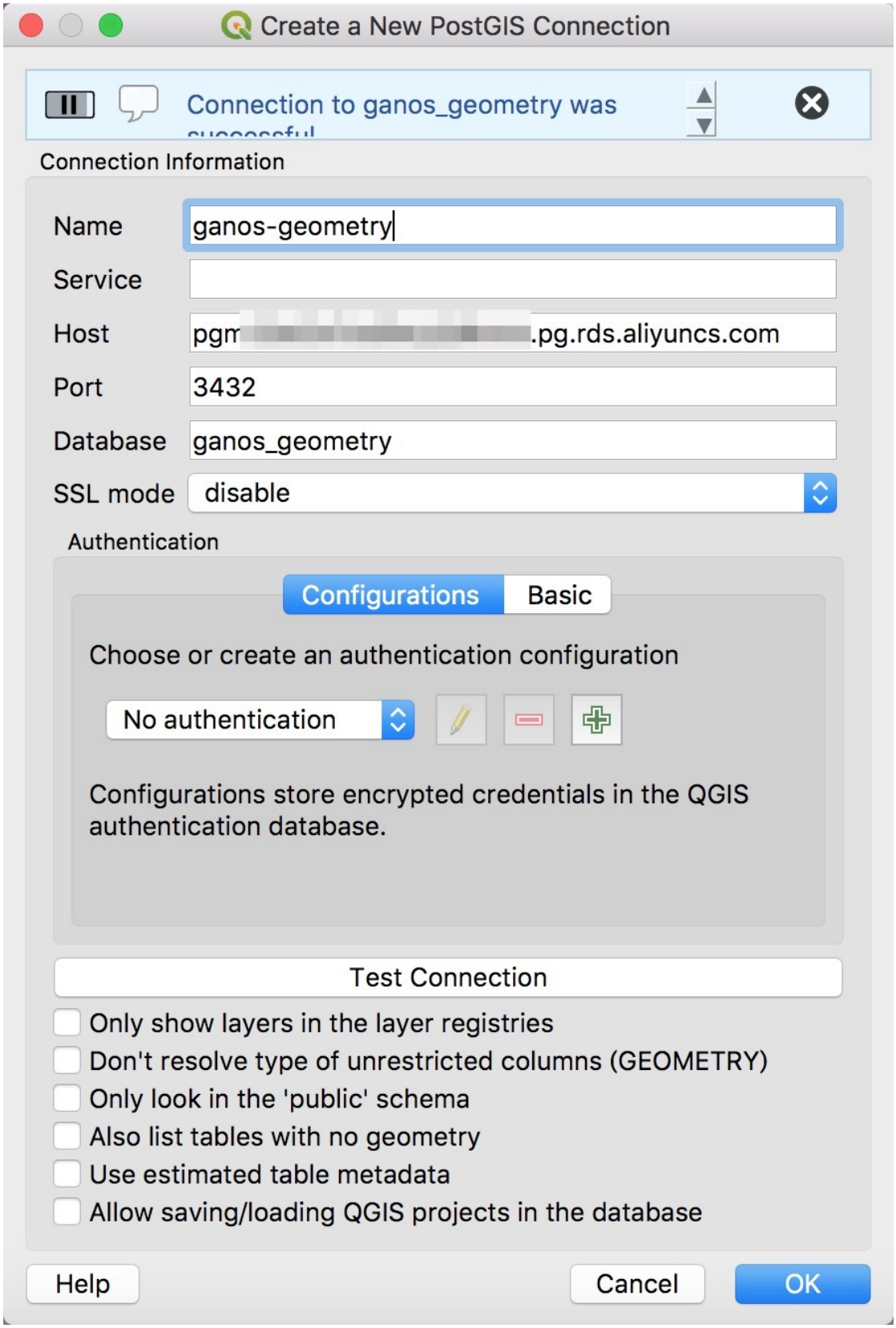
- 示例

```
ogr2ogr -nln roads -f PostgreSQL PG:"dbname='mydb' host='pgm-xxxxxxx.pg.rds.aliyuncs.com' port='3433' user='my_name' password='my_pass'" -lco SPATIAL_INDEX=GIST -lco FID=fid -overwrite "/home/roads.shp" -nlt GEOMETRY
```

QGIS桌面工具

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。

1. 创建Ganos连接。
   i. 在右侧**Brower**窗口中，直接创建PostGIS Connection。
   ii. 填入必要参数。


Create a New PostGIS Connection
Connection to ganos_geometry was
Connection Information
Name
ganos-geometry
Service
Host
.pg.rds.aliyuncs.com
Port
3432
Database
ganos_geometry
SSL mode
disable
Authentication
Configurations
Basic
Choose or create an authentication configuration
No authentication
Configurations store encrypted credentials in the QGIS authentication database.
Test Connection
Only show layers in the layer registries
Don't resolve type of unrestricted columns (GEOMETRY)
Only look in the 'public' schema
Also list tables with no geometry
Use estimated table metadata
Allow saving/loading QGIS projects in the database
Help
Cancel
OK

参数说明如下：

参数名称	描述
Name	自定义连接。
Host	数据库实例的IP地址或RDS或PolarDB外网地址，可以从aliyun控制台中获得。
Port	数据库实例的端口地址，可以从aliyun控制台中获得。
Database	需要连接的数据库名。
SSL Mode	是否使用SSL加密连接。

iii. 单击Test Connection等待连接成功。

2. 加载矢量数据。
   i. 单击菜单栏Database > DB Manager..。
   ii. 在弹出的数据库连接对话框中，选择需要导入的数据源并单击Import Layer/File...。
   iii. 在弹出的导入矢量数据对话框中，指定需要导入的矢量数据并设置导入选项，最后单击OK执行导入操作。

6.3. Trajectory常见问题

本文为您介绍Trajectory常见问题和解答，有助于您使用时空数据库。

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_them
e_json) ;
trajectory ST_append(trajectory traj, trajectory 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":"2010-01-0
1 11:30:00","end_time":"2010-01-01 12:30:00","spatial":"SRID=4326;LINESTRING(1 1,3 5)","tim
eline":["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","len
gth":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;
```

7.最佳实践

7.1. 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_e
ndtime(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。

使用分区表请参见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;
```