ALIBABA CLOUD

Alibaba Cloud

云原生关系型数据库PolarDB O引擎

Spatio-temporal Database

Document Version: 20211104

Alibaba Cloud

Legal disclaimer

Alibaba Cloud reminds you to carefully read and fully understand the terms and conditions of this legal disclaimer before you read or use this document. If you have read or used this document, it shall be deemed as your total acceptance of this legal disclaimer.

1. You shall download and obtain this document from the Alibaba Cloud website or other Alibaba Cloud-authorized channels, and use this document for your own legal business activities only. The content of this document is considered confidential information of Alibaba Cloud. You shall strictly abide by the confidentiality obligations. No part of this document shall be disclosed or provided to any third party for use without the prior written consent of Alibaba Cloud.

2. No part of this document shall be excerpted, translated, reproduced, transmitted, or disseminated by any organization, company or individual in any form or by any means without the prior written consent of Alibaba Cloud.

3. The content of this document may be changed because of product version upgrade, adjustment, or other reasons. Alibaba Cloud reserves the right to modify the content of this document without notice and an updated version of this document will be released through Alibaba Cloud-authorized channels from time to time. You should pay attention to the version changes of this document as they occur and download and obtain the most up-to-date version of this document from Alibaba Cloud-authorized channels.

4. This document serves only as a reference guide for your use of Alibaba Cloud products and services. Alibaba Cloud provides this document based on the "status quo", "being defective", and "existing functions" of its products and services. Alibaba Cloud makes every effort to provide relevant operational guidance based on existing technologies. However, Alibaba Cloud hereby makes a clear statement that it in no way guarantees the accuracy, integrity, applicability, and reliability of the content of this document, either explicitly or implicitly. Alibaba Cloud shall not take legal responsibility for any errors or lost profits incurred by any organization, company, or individual arising from download, use, or trust in this document. Alibaba Cloud shall not, under any circumstances, take responsibility for any indirect, consequential, punitive, contingent, special, or punitive damages, including lost profits arising from the use or trust in this document (even if Alibaba Cloud has been notified of the possibility of such a loss).

5. By law, all the contents in Alibaba Cloud documents, including but not limited to pictures, architecture design, page layout, and text description, are intellectual property of Alibaba Cloud and/or its affiliates. This intellectual property includes, but is not limited to, trademark rights, patent rights, copyrights, and trade secrets. No part of this document shall be used, modified, reproduced, publicly transmitted, changed, disseminated, distributed, or published without the prior written consent of Alibaba Cloud and/or its affiliates. The names owned by Alibaba Cloud shall not be used, published, or reproduced for marketing, advertising, promotion, or other purposes without the prior written consent of Alibaba Cloud. The names owned by Alibaba Cloud include, but are not limited to, "Alibaba Cloud", "Aliyun", "HiChina", and other brands of Alibaba Cloud and/or its affiliates, which appear separately or in combination, as well as the auxiliary signs and patterns of the preceding brands, or anything similar to the company names, trade names, trademarks, product or service names, domain names, patterns, logos, marks, signs, or special descriptions that third parties identify as Alibaba Cloud and/or its affiliates.

6. Please directly contact Alibaba Cloud for any errors of this document.

Document conventions

Style	Description	Example
Danger	A danger notice indicates a situation that will cause major system changes, faults, physical injuries, and other adverse results.	**Danger:** Resetting will result in the loss of user configuration data.
Warning	A warning notice indicates a situation that may cause major system changes, faults, physical injuries, and other adverse results.	**Warning:** Restarting will cause business interruption. About 10 minutes are required to restart an instance.
Notice	A caution notice indicates warning information, supplementary instructions, and other content that the user must understand.	**Notice:** If the weight is set to 0, the server no longer receives new requests.
Note	A note indicates supplemental instructions, best practices, tips, and other content.	**Note:** You can use Ctrl + A to select all files.
>	Closing angle brackets are used to indicate a multi-level menu cascade.	Click **Settings**> **Network**> **Set network type**.
Bold	Bold formatting is used for buttons , menus, page names, and other UI elements.	Click **OK**.
`Courier font`	Courier font is used for commands	Run the `cd /d C:/window` command to enter the Windows system folder.
Italic	Italic formatting is used for parameters and variables.	`bae log list --instanceid` *Instance_ID*
[] or [a|b]	This format is used for an optional value, where only one item can be selected.	`ipconfig [-all|-t]`
{} or {a|b}	This format is used for a required value, where only one item can be selected.	`switch {active|stand}`

Table of Contents

1.Overview

Spatio-temporal data is graphic and image data that contains both space and time information. Spatio-temporal data consists of multidimensional information about objects, such as the location, shape, change, and size distribution.

Description

Ganos is a spatio-temporal engine developed by Alibaba Cloud. Ganos supports a series of data types, functions, and stored procedures for Apsara PolarDB to store, index, query, analyze, and compute spatio-temporal data in a cost-effective manner.

This topic describes how to use Ganos in Apsara PolarDB to manage and analyze spatio-temporal data.

To request further assistance, you can log on to the Apsara PolarDB console. In the top navigation bar, choose **Ticket > New Ticket**.

Pricing

The Ganos engine is supported by ApsaraDB PolarDB PostgreSQL-compatible edition and PolarDB O Edition. Ganos is provided free of charge by Alibaba Cloud.

2.Models

2.1. Geometry model

Ganos Geometry is a spatial geometry extension of ApsaraDB for PolarDB. Ganos Geometry complies with OpenGIS specifications and enables ApsaraDB for PolarDB to store and manage 2D (X, Y), 3D (X, Y, Z), and 4D (X, Y, Z, M) spatial geometry data. In addition, Ganos Geometry provides a wide range of features such as spatial geometry objects, indexes, functions, and operators.

Overview

The geometry model is fully compatible with PostGIS functions and allows you to smoothly migrate existing applications.

Quick start

- Create extensions.

```
-- Create a geometry extension.
Create extension ganos_geometry cascade;
-- Create a geometry topology extension.
Create extension ganos_geometry_topology;
-- Create an SFCGAL plug-in extension.
Create extension ganos_geometry_sfcgal;
```

- Create a geometry table.

```
-- Method 1: Create a table with a geometry field.
CREATE TABLE ROADS (ID int4, ROAD_NAME varchar(25), geom geometry(LINESTRING,3857));
-- Method 2: Create a common table and then add a geometry field.
CREATE TABLE ROADS (ID int4, ROAD_NAME varchar(25));
SELECT AddGeometryColumn('roads', 'geom', 3857, 'LINESTRING', 2);
```

- Add geometry constraints.

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

- Import geometry data.

```
INSERT INTO roads (id, geom, road_name)
 VALUES (1,ST_GeomFromText('LINESTRING(191232 243118,191108 243242)',3857),'North Fifth-Ring Road
');
INSERT INTO roads (id, geom, road_name)
 VALUES (2,ST_GeomFromText('LINESTRING(189141 244158,189265 244817)',3857),'East Fifth-Ring Road');
INSERT INTO roads (id, geom, road_name)
 VALUES (3,ST_GeomFromText('LINESTRING(192783 228138,192612 229814)',3857),'South Fifth-Ring Road
');
INSERT INTO roads (id, geom, road_name)
 VALUES (4,ST_GeomFromText('LINESTRING(189412 252431,189631 259122)',3857),'West Fifth-Ring Road')
;
INSERT INTO roads (id, geom, road_name)
 VALUES (5,ST_GeomFromText('LINESTRING(190131 224148,190871 228134)',3857),'East Chang''an Avenu
e');
INSERT INTO roads (id, geom, road_name)
 VALUES (6,ST_GeomFromText('LINESTRING(198231 263418,198213 268322)',3857),'West Chang''an Avenu
e');
```

- Query geometry object information.

```
SELECT id, ST_AsText(geom) AS geom, road_name FROM roads;
-------------------------------
   id | geom                                       | road_name
--------+--------------------------------------------+-----------
    1 | LINESTRING(191232 243118,191108 243242) | North Fifth-Ring Road
    2 | LINESTRING(189141 244158,189265 244817) | East Fifth-Ring Road
    3 | LINESTRING(192783 228138,192612 229814) | South Fifth-Ring Road
    4 | LINESTRING(189412 252431,189631 259122) | West Fifth-Ring Road
    5 | LINESTRING(190131 224148,190871 228134) | East Chang'an Avenue
    6 | LINESTRING(198231 263418,198213 268322) | West Chang'an Avenue
(6 rows)
```

- Create indexes.

```
-- Create a GiST index.
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)];
-- Example
Create INDEX sp_geom_index ON ROADS USING GIST(geom);
VACUUM ANALYZE ROADS (geom);
-- Create a block range index (BRIN).
CREATE INDEX [indexname] ON [tablename] USING BRIN ([geometryfield]);
CREATE INDEX [indexname] ON [tablename] USING BRIN ([geometryfield] brin_geometry_inclusion_ops_3
d);
CREATE INDEX [indexname] ON [tablename] USING BRIN ([geometryfield] brin_geometry_inclusion_ops_4
d);
-- Create a BRIN index with a specified range.
CREATE INDEX [indexname] ON [tablename] USING BRIN ([geometryfield]) WITH (pages_per_range = [num
ber]);
```

- Access geometry objects.

```
-- Determine whether a spatial geometry object consists of only simple elements.
 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)
-- Query the largest city with roundabouts in the terrain.
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  | Anning     | 257374619.430216
(1 row)
```

- Measure and analyze spatial data, and judge spatial relationships.

```
-- Create the bc_roads table.
Create table bc_roads (gid serial, name varchar, the_geom geometry);
-- Create the bc_municipality table.
Create table bc_municipality(gid serial, code integer, name varchar, the_geom geometry);
-- Compute the length.
SELECT sum(ST_Length(the_geom))/1000 AS km_roads FROM bc_roads;
km_roads
------------------
70842.1243039643
(1 row)
-- Compute the area.
SELECT ST_Area(the_geom)/10000 AS hectares FROM bc_municipality WHERE name = 'PRINCE GEORGE';
hectares
------------------
32657.9103824927
(1 row)
-- Use the ST_Contains function.
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
```

```
-- Use the ST_Covers function.
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)
-- Use the ST_Disjoint function.
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)
-- Use the ST_Overlaps function.
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
-- Use the ST_Relate function.
SELECT ST_Relate(ST_GeometryFromText('POINT(1 2)'), ST_Buffer(ST_GeometryFromText('POINT(1 2)'),
2), '0FFFFF212');
St_relate
-----------
t
-- Use the ST_Touches function.
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)
-- Use the ST_Within function.
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)
```

- Delete extensions.

```
Drop extension ganos_geometry cascade;
```

SQL reference

For more information, see the official PostGIS reference.

2.2. Raster model

Raster data consists of a matrix of cells or pixels. The cells or pixels are organized into rows and columns to form a grid. Each cell contains a value that represents information, such as temperature.

Overview

Raster data can be digital aerial photographs, satellite images, digital images, or scanned maps.

PolarDB for PostgreSQL and PolarDB-O implement the raster model. This allows Ganos to store and analyze raster data in a cost-effective manner with the help of database technologies and methods.

Quick start

- Create an extension.

```
Create Extension Ganos_Raster cascade;
```

- Create a raster table.

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

- Import raster data from Object Storage Service (OSS).

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

- Query raster object information.

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

- Create a pyramid.

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

- Calculate the optimal pyramid level based on the world space, width, and height of a viewport.

```
Select ST_BestPyramidLevel(raster_obj, '((128.0, 30.0),(128.5, 30.5))', 800, 600) from raster_table where id = 10;
---------------------
3
```

- Retrieve a pixel matrix from the specified space of a raster object.

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

- Calculate the raster space of a clipped area.

```
Select ST_ClipDimension(raster_obj, 2, '((128.0, 30.0),(128.5, 30.5))') from rater_table where id = 10;
-----------------------------------
'((600, 720),(200, 300))'
```

- Delete the extension.

```
Drop Extension Ganos_raster cascade;
```

SQL reference

For more information, see Raster SQL reference.

2.3. Path model

Path data is a geometric network diagram that consists of edges and nodes. It is used to build road and traffic networks.

Overview

Ganos Networking is an extension of ApsaraDB for PolarDB. Ganos Networking provides a series of functions and stored procedures to find the fastest, shortest, and optimal paths based on cost models. If the cost is time, the fastest path is the optimal path. If the cost is distance, the shortest path is the optimal path. The path model can be used for path planning on a road network, path search and planning in GPS navigation on an electronic map, and routing. The path model is fully compatible with pgRouting functions and allows you to smoothly migrate existing applications.

Quick start

- Create an extension.

```
Create Extension Ganos_Networking cascade;
```

- Create a table.

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

- Insert data.

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

- Update data.

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

- Create a topology.

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

- Query the shortest path.

```
-- Use Dijkstra's algorithm to query the shortest path.
SELECT * FROM pgr_dijkstra(
  'SELECT id, source, target, cost, reverse_cost FROM edge_table',
  2, 3
);
 seq | path_seq | node | edge | cost | agg_cost
-----+----------+------+------+------+----------
   1 |        1 |    2 |    4 |    1 |        0
   2 |        2 |    5 |    8 |    1 |        1
   3 |        3 |    6 |    9 |    1 |        2
   4 |        4 |    9 |   16 |    1 |        3
   5 |        5 |    4 |    3 |    1 |        4
   6 |        6 |    3 |   -1 |    0 |        5
(6 rows)
-- Use the A* algorithm to query the shortest path.
SELECT * FROM pgr_astar(
  'SELECT id, source, target, cost, reverse_cost, x1, y1, x2, y2 FROM edge_table',
  2, 12,
  directed := false, heuristic := 2);
 seq | path_seq | node | edge | cost | agg_cost
-----+----------+------+------+------+----------
   1 |        1 |    2 |    2 |    1 |        0
   2 |        2 |    3 |    3 |    1 |        1
   3 |        3 |    4 |   16 |    1 |        2
   4 |        4 |    9 |   15 |    1 |        3
   5 |        5 |   12 |   -1 |    0 |        4
(5 rows)
-- Use the turn restricted shortest path (TRSP) algorithm to query the shortest path.
SELECT * FROM pgr_trsp(
    'SELECT id::INTEGER, source::INTEGER, target::INTEGER, cost FROM edge_table',
    2, 7, false, false,
    'SELECT to_cost, target_id::int4,
    from_edge || coalesce('','' || via_path, '''') AS via_path
    FROM restrictions'
  );
 seq | id1 | id2 | cost
-----+-----+-----+------
   0 |   2 |   4 |    1
   1 |   5 |  10 |    1
   2 |  10 |  12 |    1
   3 |  11 |  11 |    1
   4 |   6 |   8 |    1
   5 |   5 |   7 |    1
   6 |   8 |   6 |    1
   7 |   7 |  -1 |    0
(8 rows)
```

- Delete the extension.

```
Drop Extension Ganos_Networking cascade;
```

SQL reference

For more information, see the official pgRouting manual.

2.4. Point cloud model

Point cloud data consists of the points generated by a 3D scanner when it scans an object. Each point is represented by a set of 3D coordinates, and some may contain RGB or intensity information. Point cloud data contains spatial coordinate information, and has a large number of points and complex attribute dimensions.

Overview

Ganos PointCloud is an extension of ApsaraDB for PolarDB. It enables ApsaraDB for PolarDB to store and manage point cloud data efficiently and fast. Ganos PointCloud provides features such as point cloud data compression, decompression, and attribute statistics collection. It can also work with Ganos Geometry to support spatial analysis of point cloud data.

Point cloud data types

Point cloud data types include pcpoint and pcpatch. For the pcpoint type, each point is stored as a row. The dimensions of a point are defined in the point cloud schema. For the pcpatch type, points are stored as a collection. Point cloud data of the pcpatch type can be compressed to reduce storage space and supports spatial queries. The compression method is specified by the compression parameter in the point cloud schema.

Point cloud schema

The pointcloud_formats table stores point cloud schemata. Each schema lists the attribute dimensions of a point cloud object and provides the information about each dimension, such as the size, type, name, and description.

Quick start

- Create extensions.

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

- Insert a point cloud schema.

```
INSERT INTO pointcloud_formats (pcid, srid, schema) VALUES (1, 4326,
'<? xml version="1.0" encoding="UTF-8"?>
<pc:PointCloudSchema xmlns:pc="http://pointcloud.org/schemas/PC/1.1"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <pc:dimension>
  <pc:position>1</pc:position>
  <pc:size>4</pc:size>
  <pc:description>X coordinate as a long integer. You must use the
          scale and offset information of the header to
          determine the double value.</pc:description>
  <pc:name>X</pc:name>
  <pc:interpretation>int32_t</pc:interpretation>
  <pc:scale>0.01</pc:scale>
 </pc:dimension>
 <pc:dimension>
  <pc:position>2</pc:position>
  <pc:size>4</pc:size>
  <pc:description>Y coordinate as a long integer. You must use the
          scale and offset information of the header to
          determine the double value.</pc:description>
  <pc:name>Y</pc:name>
  <pc:interpretation>int32_t</pc:interpretation>
  <pc:scale>0.01</pc:scale>
 </pc:dimension>
 <pc:dimension>
  <pc:position>3</pc:position>
  <pc:size>4</pc:size>
  <pc:description>Z coordinate as a long integer. You must use the
          scale and offset information of the header to
          determine the double value.</pc:description>
  <pc:name>Z</pc:name>
  <pc:interpretation>int32_t</pc:interpretation>
  <pc:scale>0.01</pc:scale>
 </pc:dimension>
 <pc:dimension>
  <pc:position>4</pc:position>
  <pc:size>2</pc:size>
  <pc:description>The intensity value is the integer representation
          of the pulse return magnitude. This value is optional
          and system-specific. However, it should always be
          included if available.</pc:description>
  <pc:name>Intensity</pc:name>
  <pc:interpretation>uint16_t</pc:interpretation>
  <pc:scale>1</pc:scale>
 </pc:dimension>
 <pc:metadata>
  <Metadata name="compression">dimensional</Metadata>
 </pc:metadata>
</pc:PointCloudSchema>');
```

- Create a point cloud table.

```
-- Use the pcpoint data type.
CREATE TABLE points (
  id SERIAL PRIMARY KEY,
  pt PCPOINT(1)
);
-- Use the pcpatch data type.
CREATE TABLE patches (
  id SERIAL PRIMARY KEY,
  pa PCPATCH(1)
);
```

- Insert pcpoint data.

```
INSERT INTO points (pt)
SELECT ST_MakePoint(1, ARRAY[x,y,z,intensity])
FROM (
 SELECT
 -127+a/100.0 AS x,
  45+a/100.0 AS y,
     1.0*a AS z,
      a/10 AS intensity
 FROM generate_series(1,100) AS a
) AS values;
SELECT ST_MakePoint(1, ARRAY[-127, 45, 124.0, 4.0]);
--------------------------
010100000064CEFFFF94110000703000000400
SELECT ST_AsText('010100000064CEFFFF94110000703000000400'::pcpoint);
--------------------------
{"pcid":1,"pt":[-127,45,124,4]}
```

- Insert pcpatch data.

```
INSERT INTO patches (pa)
SELECT ST_Patch(pt) FROM points GROUP BY id/10;
SELECT ST_AsText(ST_MakePatch(1, ARRAY[-126.99,45.01,1,0, -126.98,45.02,2,0, -126.97,45.03,3,0]));
--------------------------
{"pcid":1,"pts":[
 [-126.99,45.01,1,0],[-126.98,45.02,2,0],[-126.97,45.03,3,0]
]}
```

- Compute the average values of all attribute dimensions in a pcpatch object.

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

- Delete extensions.

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

SQL reference

For more information, see Point cloud SQL reference.

2.5. Trajectory model

Trajectory data records the continuous location update information about a moving feature, such as a vehicle or a person. Trajectory data is a type of typical spatio-temporal data. You can analyze and use trajectory data to study many important issues.

Overview

Ganos Trajectory is an extension of PolarDB. Ganos Trajectory provides a series of data types, functions, and stored procedures. This allows you to manage, query, and analyze spatio-temporal trajectory data.

Quick start

- Create the Ganos Trajectory extension.

```
Create Extension Ganos_trajectory cascade;
```

- Create the enumeration type for a trajectory.

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

- Create a trajectory table.

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

- Insert trajectory data records into the trajectory table.

```
insert into traj_table values
(1, ST_MakeTrajectory('STPOINT'::leaftype, st_geomfromtext('LINESTRING (114 35, 115 36, 116 37)', 4326),
'[2010-01-01 14:30, 2010-01-01 15:30)'::tsrange, '{"leafcount": 3,"attributes" : {"velocity" : {"type":"integer
","length":4,"nullable":false,"value":[120, 130, 140]},"accuracy":{"type":"integer","length":4,"nullable":fa
lse,"value":[120, 130, 140]},"bearing":{"type":"float","length":4,"nullable":false,"value":[120, 130, 140]},"a
cceleration":{"type":"float","length":4,"nullable":false,"value":[120, 130, 140]}}}')),
(2, ST_MakeTrajectory('STPOINT'::leaftype, st_geomfromtext('LINESTRING (114 35, 115 36, 116 37)', 4326),
'2010-01-01 14:30'::timestamp, '2010-01-01 15:30'::timestamp, '{"leafcount": 3,"attributes" : {"velocity" : {"
type":"integer","length":4,"nullable":false,"value":[120, 130, 140]},"accuracy":{"type":"integer","length":
4,"nullable":false,"value":[120, 130, 140]},"bearing":{"type":"float","length":4,"nullable":false,"value":[12
0, 130, 140]},"acceleration":{"type":"float","length":4,"nullable":false,"value":[120, 130, 140]}}}')),
(3, ST_MakeTrajectory('STPOINT'::leaftype, st_geomfromtext('LINESTRING (114 35, 115 36, 116 37)', 4326),
ARRAY['2010-01-01 14:30'::timestamp, '2010-01-01 15:00'::timestamp, '2010-01-01 15:30'::timestamp], '{"le
afcount": 3,"attributes" : {"velocity" : {"type":"integer","length":4,"nullable":false,"value":[120, 130, 140]}
,"accuracy":{"type":"integer","length":4,"nullable":false,"value":[120, 130, 140]},"bearing":{"type":"float"
,"length":4,"nullable":false,"value":[120, 130, 140]},"acceleration":{"type":"float","length":4,"nullable":fa
lse,"value":[120, 130, 140]}}}')),
(4, ST_MakeTrajectory('STPOINT'::leaftype, st_geomfromtext('LINESTRING (114 35, 115 36, 116 37)', 4326),
'[2010-01-01 14:30, 2010-01-01 15:30)'::tsrange, null));
```

- Create a trajectory index on the trajectory table.

```
--Create a trajectory index to accelerate the process of filtering spatio-temporal data records.
create index tr_index on trajtab using trajgist (traj);
--Run spatial data queries. You can find that the trajectory index accelerates the process of filtering spatia
l data records.
select id, traj_id from traj_test where st_3dintersects(traj, ST_GeomFromText('POLYGON((116.467478518
05917 39.92317964155052,116.4986540687358 39.92317964155052,116.4986540687358 39.9445240171151
6,116.46747851805917 39.94452401711516,116.46747851805917 39.92317964155052))'));
--Run temporal data queries. You can find that the trajectory index accelerates the process of filtering tem
poral data records.
select id, traj_id from traj_text where st_TWithin(traj, ST_ToBox('2008-02-02 13:30:44'::timestamp,'2008-0
2-03 17:30:44'::timestamp));
--Run spatio-temporal data queries. You can find that the trajectory index accelerates the process of filter
ing spatio-temporal data records.
select id, traj_id from traj_test where st_3dintersects(traj, ST_GeomFromText('POLYGON((116.467478518
05917 39.92317964155052,116.4986540687358 39.92317964155052,116.4986540687358 39.9445240171151
6,116.46747851805917 39.94452401711516,116.46747851805917 39.92317964155052))'),'2008-02-02 13:30:
44'::timestamp,'2008-02-03 17:30:44'::timestamp);
```

- Create a trajectory index in a specific dimension.

```
--If you want to analyze trajectory data only from a specific dimension, create a trajectory index in that di
mension. For example, if you do not want to analyze trajectory data in the z dimension, create a two-dime
nsional temporal trajectory index by using trajgist_op_2dt.
create index tr_timespan_time_index on trajtab using trajgist (traj trajgist_op_2dt);
--Query two-dimensional temporal data. The queries are run at a fast rate.
select id, traj_id from traj_test where st_2dintersects(traj, ST_GeomFromText('POLYGON((116.467478518
05917 39.92317964155052,116.4986540687358 39.92317964155052,116.4986540687358 39.9445240171151
6,116.46747851805917 39.94452401711516,116.46747851805917 39.92317964155052))'),'2008-02-02 13:30:
44'::timestamp,'2008-02-03 17:30:44'::timestamp);
--Create more trajectory indexes based on your business requirements. If you create one trajectory index,
all queries run based on the trajectory index. If you create multiple trajectory indexes, ApsaraDB for RDS s
elects the optimal trajectory index when you run queries.
create index tr_timespan_time_index on trajtab using trajgist (traj trajgist_op_2d);
select id, traj_id from traj_test where st_2dintersects(traj, ST_GeomFromText('POLYGON((116.467478518
05917 39.92317964155052,116.4986540687358 39.92317964155052,116.4986540687358 39.9445240171151
6,116.46747851805917 39.94452401711516,116.46747851805917 39.92317964155052))'));
```

- Query the start time and end time of trajectories.

```
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)
```

- Query the trajectories of moving objects.

```
--Query the attributes of a trajectory point by using an interpolation function.
Select ST_velocityAtTime(traj, '2010-01-01 12:45') from traj_table  where id > 5;
st_velocityattime
-------------------
5
5
4.16666666666667
(3 rows)
```

- Analyze the proximity among trajectories.

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

- Delete the Ganos Trajectory extension.

```
Drop Extension Ganos_trajectory cascade;
```

SQL reference

For more information, see Trajectory SQL reference.

2.6. Grid model

Grid models are discrete, multi-scale location identification systems that are developed from GeoSOT.

Overview

A grid model allows you to assign globally unique codes to various objects within the earth space (from the earth's core to the earth's surface). The globally unique grid code of an object can be used to associate the object with various data that is collected from the same location as the object.

Quick start

- Create an extension.

```
CREATE EXTENSION Ganos_GeomGrid CASCADE;
```

- Create a table with a grid code.

```
CREATE TABLE t_grid(id integer,
            geom geometry,  -- A geometry.
            grid1 geomgrid[], -- A grid code with a precision of 1.
            grid2 geomgrid[], -- A grid code with a precision of 2.
            grid3 geomgrid[] -- A grid code with a precision of 3.
           );
```

- Compute the grid code of the table.

```
--Insert data into the table.
INSERT INTO t_grid(id, geom)
VALUES (1, ST_GeomFromText('POINT(116.31522216796875 39.910277777777778)', 4490)),
    (2, ST_GeomFromText('POINT(116.31522217796875 39.910277776777778)', 4490)),
    (3, ST_GeomFromText('POINT(116.31522217797875 39.910277776787778)', 4490)),
    (4, ST_GeomFromText('POINT(116.31522227796875 39.910277776775778)', 4490));
--Create grid codes of different precision levels.
UPDATE t_grid
SET grid1 = ST_AsGrid(geom, 10),
  grid2 = ST_AsGrid(geom, 15),
  grid3 = ST_AsGrid(geom, 26);
```

- Create indexes on grid codes.

```
--Create GIN indexes on grid codes of different precision levels.
CREATE INDEX idx_grid_gin1
ON t_grid
USING GIN(grid1);
CREATE INDEX idx_grid_gin2
ON t_grid
USING GIN(grid2);
CREATE INDEX idx_grid_gin3
ON t_grid
USING GIN(grid3);
```

- Query data.

```
--Query data that resides in a grid.
SELECT id
FROM t_grid
WHERE grid2 = ARRAY[ST_GridFromText('G001310322230230')];
--Query data that intersects with a grid.
SELECT id
FROM t_grid
WHERE grid3 @> ARRAY[ST_GridFromText('G00131032223023031031033223')];
--Query data that intersects with more than one grid.
SELECT id
FROM t_grid
WHERE grid3 && ARRAY[ST_GridFromText('G00131032223023031031211001'),
                ST_GridFromText('G00131032223023031031211111')];
--Query data that intersects with more than one geometry in a grid.
SELECT id
FROM t_grid
WHERE grid3 &&
ST_AsGrid(
 ST_GeomFromText('LINESTRING(116.31522216796875 39.910277777777778, 116.31522217797875 39.910
277776787778)', 4490), 26);
```

- Delete the extension that you created.

```
DROP EXTENSION Ganos_GeomGrid CASCADE;
```

2.7. Vector pyramid

Vector pyramids are structures that are designed to display tens of millions of spatial geometry data records.

Overview

Vector pyramids allow you to create sparse indexes on spatial geometry data records and to preprocess data-intensive areas based on the specified rules. This way, you can obtain standard data records in the MVT and PBF formats.

Vector pyramids provided by Ganos allow you to preprocess hundreds of millions of spatial geometry data records in minutes and to display the preprocessing results in seconds.

Quick start

- Create an extension.

```
CREATE EXTENSION ganos_geometry_pyramid CASCADE;
```

- Create a pyramid for a spatial table.

```
--Create a pyramid for the data table named test and specify the names of element ID fields. The names m
ust be of the int4 or int8 data type.
--Specify the name of the spatial fields in the test table. Before you specify the name of a spatial field, you
must create a spatial index for the field.
SELECT ST_BuildPyramid('test', 'geom', 'id', '');
```

- Read MVT-formatted data records from the pyramid that you created.

```
--Read the data record with the tile ID 0_0_0 from the pyramid. You can specify any tile ID. The system ret
urns a data record regardless of whether the specified tile ID can be found in the pyramid.
--Make sure that the specified tile ID follows the z_x_y format and the EPSG:3857 coordinate system.
SELECT ST_Tile('test', '0_0_0');
```

- Delete the pyramid that you created.

```
--Delete the pyramid named test.
SELECT ST_DeletePyramid('test');
```

- Delete the extension that you created.

```
DROP EXTENSION ganos_geometry_pyramid CASCADE;
```

Parameter settings

- Pyramid name

  The default name of a pyramid is the same as the data table to which the pyramid belongs. You can change the default name to map one data table to multiple pyramids.

```
--Create a pyramid named hello for the data table named test.
SELECT ST_BuildPyramid('test', 'geom', 'id', '{"name": "hello"}');
```

- Number of parallel tasks

  You can specify the maximum number of tasks that can run in parallel to create pyramids. The default value is 0. The value 0 indicates that the number of parallel tasks is not limited. The value cannot exceed four times the number of cores that are configured.

Pyramids are created in parallel in compliance with the two-phase commit (2PC) protocol. You must set the max_prepared_transactions parameter to a value that is greater than or equal to 100. After you set the parameter, you must restart your PolarDB cluster to make the new parameter setting take effect.

```
--Configure four cores to create pyramids in parallel.
SELECT ST_BuildPyramid('test', 'geom', 'id','{"parallel": 4}');
```

- Tile parameters

  You can specify the tile size and the tile extension size.

  - Valid tile sizes are multiples of 256 within the range of 0 to 4096.
  - Valid tile extension sizes are within the range of 0 to 256.

```
--Set the tile size to 512 and the tile extension size to 8.
SELECT ST_BuildPyramid('test', 'geom', 'id','{
                "tileSize": 512,
                "tileExtend": 8
               }');
```

- Maximum number of pyramid layers

  You can specify the maximum number of layers in a pyramid. The default value is 16. If the number of zoom levels is greater than the maximum number of pyramid layers, the layers beyond the maximum number are no longer included in the pyramid. If you do not set this parameter, the system provides a maximum number based on data density.

```
--Set the maximum number of pyramid layers to 12. If the number of zoom levels exceeds 12, the system r
eads data in real time to generate data records that are in the MVT format.
SELECT ST_BuildPyramid('test', 'geom', 'id', '{"maxLevel": 12}');
```

- Layered processing

  You can specify unique criteria that are used to process each layer of a pyramid. The criteria include the fields that you want to display and the filter conditions.

  You can specify the criteria by using the `buildRules` element.

  The system requires a long period of time to generate the top layer of a pyramid. You can use layered processing rules to skip the top layer.

```
--Enable layered processing.
--Specify only the "1!=1" filter condition for Layers 0 through 5.The system generates empty data records in the MVT format and does not display the data of Layers 0 through 5.
--Specify the name field that you want to display and the "code = 1" filter condition for Layers 6 through 9.
--Specify only the name and width fields that you want to display for Layers 10 through 15.
SELECT ST_BuildPyramid('test', 'geom', 'id', '{
                "buildRules":[
                 {
                  "level":[0,1,2,3,4,5],
                  "value": {
                   "filter": "1 != 1"
                  }
                 },
                 {
                  "level":[6,7,8,9],
                  "value": {
                   "filter": "code = 1",
                   "attrFields": ["name"]
                  }
                 },
                 {
                  "level":[10,11,12,13,14,15],
                  "value": {
                   "attrFields": ["name", "width"]
                  }
                 }
                ]
               }');
```

Advanced features

You can merge the data records within a specific tile based on attributes to reduce the total number of data records.

```
--Separately merge the data records whose codes are 1 and those whose codes are 2.
SELECT ST_BuildPyramid('test', 'geom', 'id', '{
                "buildRules":[
                    {
                      "level":[0,1,2,3,4,5],
                      "value": {
                        "merge": ["code=1","code=2"]
                      }
                    }
                  ]
                }');
```

3.Advanced usage

3.1. Create spatial indexes in parallel

This feature uses the GiST Sort method to parallelize the index creation process, which significantly reduces the read/write operations on disks and accelerates index creation.

Precautions

The GiST Sort method is only applicable to points. Using this method for other types of spatial data will affect the query performance of indexes.

Usage

> **Note** The number of workers and the total memory usage of workers in the following statements are provided for reference. You can configure them as needed.

1. Enable the GiST Sort feature.

   ```
   set polar_enable_gist_sort=on;
   ```

2. Configure the number of parallel workers.

   The more parallel workers that you use to scan data tables, the higher the CPU load is during query. Therefore, we recommend that the number of workers you set do not exceed the number of physical CPU cores.

   ```
   set max_parallel_maintenance_workers=4;
   ```

3. Configure the total memory usage of parallel workers. This parameter is recommended to be at least 1G.

   ```
   set maintenance_work_mem='1GB';
   ```

4. Create a GiST index.

   ```
   create index on t using gist(geom);
   ```

5. Execute the following statement to disable the Gist Sort feature:

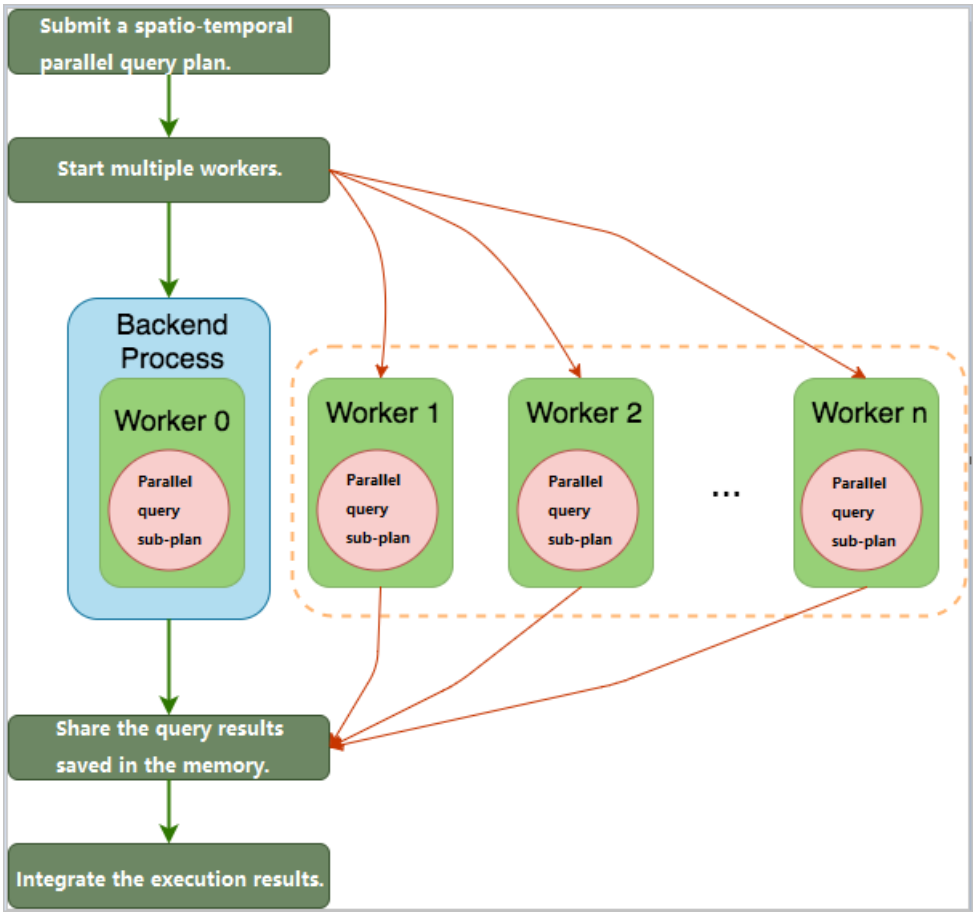
   ```
   set polar_enable_gist_sort=off;
   ```

3.2. Enable spatio-temporal parallel query

Ganos can use the parallel query capability of PostgreSQL to accelerate complex spatio-temporal data queries that involve a large amount of data.

Parallel query principles

Parallel query is performed at the table level. The following figure shows the parallel query process.



Precautions

- A larger number of workers used for parallel query indicate a higher CPU load. If the CPU load is already high, we recommend that you set the number of workers to 2 by setting the max_parallel_workers_per_gather parameter to 2.
- When enabling parallel query for highly concurrent access requests on a server with limited memory, you must properly set the work_mem parameter (to a minimum of 64 KB). You must ensure that the number of concurrent access requests multiplied by the number of parallel workers multiplied by the value of the work_mem parameter does not exceed 60% of the server memory.

Usage

To enable Ganos parallel query, perform the following operations:

1. Set parallel query parameters in the postgresql.conf profile.
   - Set the max_parallel_workers parameter to specify the total number of parallel workers that can be enabled, in the range of 8 to 32. The value of this parameter must be smaller than that of the max_worker_processes parameter.
   - Set the max_parallel_workers_per_gather parameter to specify the maximum number of parallel workers for a single query gather, in the range of 2 to 4. The value of this parameter must be smaller than that of the max_parallel_workers parameter.
   - To forcibly enable parallel query, set the force_parallel_mode parameter to on.
   - To set the number of parallel workers for a table, execute the following SQL statement: alter

table table_name set (parallel_workers=n).

> **Note** In the preceding SQL statement, n indicates the number of parallel workers. For more information about how to set an appropriate value for n, see the description of the max_parallel_workers_per_gather parameter.

2. Increase the cost of Ganos functions.

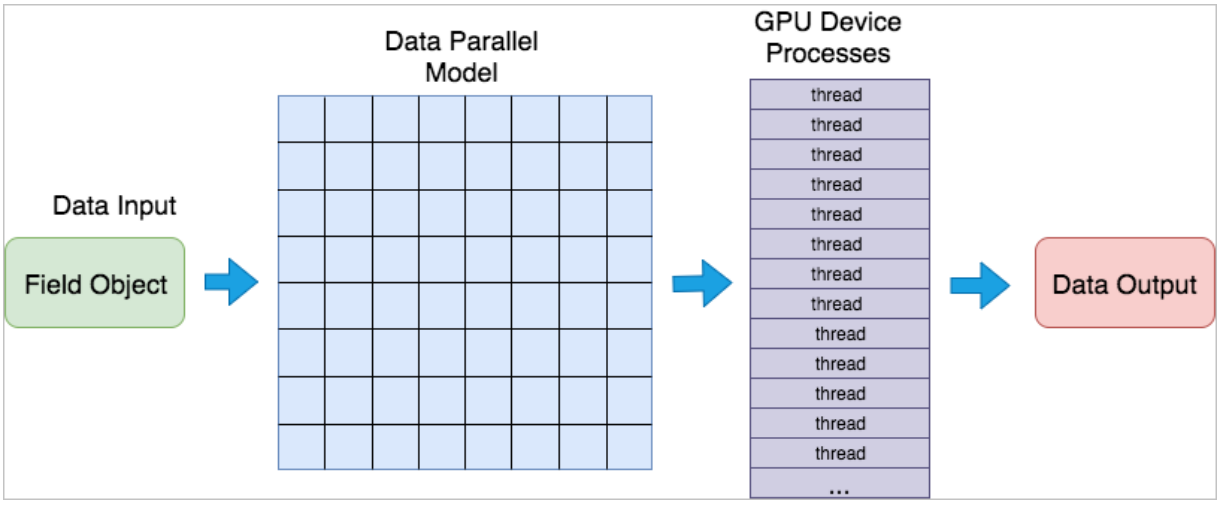
   After a Ganos module extension is created, functions have a default cost. If the data size of the table for the module extension is small, parallel query is disabled by default. If a function is computing-intensive and parallel query is suitable for the function, you must increase the cost of the function before you can enable parallel query.

3.3. Enable GPU-accelerated computing

Due to its special hardware architecture, a GPU has great advantages over a CPU in processing computing-intensive and easy-to-parallel code.

Acceleration principles

In a database, GPU parallel acceleration is performed at the object level. Ganos converts a field object to a model suitable for parallel computing, and uses the multiple cores of the GPU for parallel computing. The following figure shows the parallel computing process.



Notes

A single GPU has limited resources. Therefore, we recommend that you disable GPU-accelerated computing in a session for high-concurrency scenarios.

Procedure

When detecting a GPU, Ganos automatically enables GPU-accelerated computing. You do not need to set any parameters. GPU-accelerated computing is transparent and imperceptible to you. In addition, Ganos allows you to enable or disable GPU-accelerated computing at the session level.

1. Check whether a GPU is available in the current environment.
   i. Run the `create extension ganos_raster cascade` statement to create a Ganos Raster extension.

ii. Run the `select st_checkgpu()` statement.

- If Ganos detects a GPU in the current environment, it returns the GPU information.

```
rasterdb=# select st_checkgpu();
            st_checkgpu
-----------------------------------------------------------------------------------------------------------------------------
 [GPU(0) prop]multiProcessorCount=20; sharedMemPerBlock=49152; totalGlobalMem=-608239616; maxThreadsPerBlock=1024; maxThreadsPerMultiProcessor=2048; cudaThreadGetLimit=1024.
(1 row)
```

- If Ganos does not detect a GPU in the current environment, it returns an error message.

```
rasterdb=# select st_checkgpu();
            st_checkgpu
-----------------------------------------------------------------------------------------------------------------------------
--------
There is not gpu device on current environment, cuda_errorcode=35, errormsg=CUDA driver version is insufficient for CUDA runtime version.
(1 row)
```

> **Note** GPU-accelerated computing can be enabled only in an environment where a GPU is available.

2. Enable or disable GPU-accelerated computing at the session level.
   - In an environment with a GPU, Ganos enables GPU-accelerated computing by default. To disable GPU-accelerated computing to use the original CPU computing mode, perform the following operation in a session:

     Run the set ganos.raster.use_cuda=off statement.

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

   - To enable GPU-accelerated computing again, run the set ganos.raster.use_cuda=on statement.

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

3. Implement GPU-accelerated computing in a Ganos module.

> **Note** Currently, you can implement GPU-accelerated computing only in the Ganos Raster module. GPU-accelerated computing will be supported by the Ganos Trajectory and Ganos Geometry modules in the future.

3.4. Spatial-temporal object storage optimization

3.4.1. Feature signature-based storage optimization for large spatio-temporal objects

The feature signatures of large spatio-temporal objects are used in Ganos to optimize the storage of these objects. This storage optimization feature allows you to store the metadata of these objects in-row and store the entity data of these objects off-row. This increases processing performance and reduces I/O overhead in scenarios such as spatio-temporal index creation and spatial joins.

Configure the storage optimization feature

After you connect to your PolarDB cluster, you can enable or disable the storage optimization feature by using a Grand Unified Configuration (GUC) parameter. By default, the storage optimization feature is disabled.

- To enable the storage optimization feature, run the following command:

```
set polar_enable_storage_partial = true;
```

- To disable the storage optimization feature, run the following command:

```
set polar_enable_storage_partial = false;
```

Configure the threshold for storage optimization

After you enable the storage optimization feature, PolarDB implements in-row storage and off-row storage on a large spatio-temporal object only when the number of bytes for the object exceeds the value of the `polar_partial_external_threshold` parameter.

The value of the `polar_partial_external_threshold` parameter ranges from 1000 to 8000. The default value is 2000. Before you create a table, you can reconfigure this parameter. Example:

```
set polar_partial_external_threshold = 3000;
```

3.4.2. ST_SetTypeStorage

This topic describes the ST_SetTypeStorage function, which is used to set the storage mode for specific data types. This function allows you to store some types of data in-row and store the other types of data off-row.

Syntax

```
bool ST_SetTypeStorage(cstring typeName, cstring storageStrategy, integer size);
```

Parameters

Parameter	Description
typeName	The types of data whose storage mode you want to specify. The supported data types include geometry and geography.
storageStrategy	The storage mode that you want to specify. Set the value to main.
size	The number of bytes that can be stored in-row.

Description

If the size per object for a specific data type is large, you can store the metadata in-row and store the entity data off-row. For normal query and analysis workloads, PolarDB needs to read only the in-row metadata. This reduces the I/O operations on the disk.

The size parameter specifies the number of in-row bytes. If you set this parameter to 0, data is still stored in main mode. The following table lists the recommended values of this parameter for various data types.

Data type	Dimension	Recommended value
geometry	2d(x,y)	24
	3d(x,y,z) and 3dm(x,y,m)	32
	4d(x,y,z,m)	40
geography	2d(x,y)	32

Example

```
Select ST_SetTypeStorage('geometry','main', 41);
 st_settypestorage
-------------------
 t
(1 row)
```

3.4.3. Use the Simple storage policy

This topic describes how to use the Simple storage policy in the spatio-temporal database Ganos.

Background information

Spatio-temporal data features large data objects and the total amount of spatio-temporal data is large. In most cases, data of the types, such as grids, trajectories, and point clouds, in Ganos must be compressed by using professional compression algorithms and then stored in database tables. The storage policy provided by the current database supports only fixed and simple compression algorithms. Data can also be stored in external tables without being compressed. Due to this policy, spatio-temporal occupies a lot of storage space and affects query performance.

Therefore, a simple storage policy called Simple is provided in Ganos. This policy allows you to use a custom compression algorithm to compress data before the data is stored. The compressed data is stored in basic tables if possible. The policy reduces storage costs, and the impact on query performance as much as possible.

Use the policy

Specify the Grand Unified Configuration (GUC) parameter. By default, this parameter is enabled.

- To enable the GUC parameter, execute the following statement:

```
SET polar_enable_storage_simple=true;
```

- To disable the GUC parameter, execute the following statement:

```
SET polar_enable_storage_simple=false;
```

Set the specified field of the table to `storage simple`, as shown in the following example:

```
ALTER TABLE tbname ALTER colname SET storage simple;
```

After the settings are complete, the colname field is automatically stored based on the Simple storage policy.

4.Raster SQL reference

4.1. Basic concepts

This topic introduces the basic concepts related to Raster SQL.

Term	Description
raster object	The raster that is short for a raster object. It is regular grids into which space is divided. Each grid is called a cell. Each cell is assigned an attribute value to represent a data model of an entity. A raster object can be a satellite image, digital elevation model (DEM), or picture.
cell/pixel	The raster cell. It is also called a pixel. This indicates that a cell is a grid in a raster object. Each cell can have different data types, such as Byte, Short, Int, Double.
band	The raster band. A band is a single matrix of cell values in a raster object. A raster object can have multiple bands.
chunk	The raster chunk. You can customize the size of a chunk, such as 256 × 256 × 3.
pyramid	The raster pyramid. The downsampled version of a source raster object. A pyramid can contain multiple downsampled layers. Consecutive pyramid layers are downsampled at a scale of 2:1. Layer 0 stores the raw data.
pyramid level	The layer of a raster pyramid.
mosaic	The operation to integrate multiple raster objects into an existing raster dataset.
interleaving	The interleaving method of pixels in a raster. The interleaving methods are band sequential (BSQ), band interleaved by pixel (BIP), and band interleaved by line (BIL).
world space	The world coordinate space. It indicates the geographic coordinate space of a raster object.
raster space	The coordinate space of a raster. It indicates the pixel coordinate space of a raster object. The **upper-left corner** of the raster is used as the starting point.
metadata	The metadata of a raster, including the spatial range, projection type, and pixel type. The metadata of the remote sensing platform is excluded.

4.2. Parallel operations

Ganos uses multiple CPUs to increase the performance of queries or computing. This feature is called parallel operation. Ganos supports parallel SQL statements and parallel raster-level operations.

Execute SQL statements in parallel

- How it works

PostgreSQL uses multiple CPUs to generate multiple query plans in parallel. Then, PostgreSQL balances the new query plans across these CPUs. This increases the query performance.

- Scenario

  If you want to scan a large number of raster objects to identify those that meet the specified spatial scope or attribute requirements, we recommend that you execute parallel SQL statements. This significantly reduces the time that is required to run queries.

- Scope

  All the read-only functions that are used to compute raster objects in Ganos support parallel queries. These functions include the function that is used to query the attributes of a raster object.

Manage raster objects in parallel

- How it works

  In most cases, Ganos uses multiple CPUs to perform multiple operations in parallel on the subsets of a raster object. This reduces the time that is required for the overall computation on the raster object. Each subset is independently computed. When the computations on all the subsets are complete, the parallel operations on the raster object end.

- Scenario

  The raster object that you want to compute is large. In this case, a long period of time is required to complete the overall computation.

- Scope

  For more information about the functions that support parallel raster-level operations, see the following topics:

  - ST_ImportFrom
  - ST_BuildPyramid
  - ST_RPCRectify

Before you start

- Set the maximum number of transactions that can be used in the prepared state.

  You can use the max_prepared_transactions parameter to set the maximum number. The default value of this parameter is 0. We recommend that you set this parameter to the value of the max_connections parameter. This ensures that one transaction in the Prepared state is assigned to each connection.

  > **Note** After you reconfigure this parameter, you must restart your instance.

- Set the degree of parallelism

  Before you invoke a function that supports parallel raster-level operations, you must set the degree of parallelism. By default, the value of a Grand Unified Configuration (GUC) parameter is used as the degree of parallelism. This applies if you do not specify the degree of parallelism or set it to 0. The GUC parameter is set to ganos.parallel.degree. The default value of the ganos.parallel.degree is 1. The value 1 indicates that the function does not support parallel raster-level operations.

  You can also set this parameter to a positive integer for a function that supports parallel raster-level operations. Then, you can split the computation on a raster object into parallel tasks. Example:

```
select ST_ImportFrom('chunk_table','OSS://<akxxxx>:<ak_secretxxxx>@oss-cn-beijing-internal.aliyuncs.com/mybucket/data/image.nc:hcc', '{}', '{"parallel": 4}');
```

- Set the transactional consistency

  You can specify the transactional consistency by using the ganos.parallel.transaction GUC parameter. Valid values:

  - **transaction_commit** specifies that Ganos can commit or roll back parallel transactions based on the main transaction. This is the default value.
  - **fast_commit**: specifies that Ganos cannot commit or roll back parallel transactions based on the main transaction.

> Note
>
> - To ensure the optimal performance, you can use the ST_CreateChunkTable function to create a chunk table in advance.
> - If a parallel function cannot return a temporary chunk table that is created by an anonymous user, you must first create a chunk table. Then, specify the chunk table in the chunktable parameter.

4.3. Raster creation

4.3.1. ST_CreateRast

This topic describes the ST_CreateRast function, which creates a raster object that is based on an Alibaba Cloud Object Storage Service (OSS) object.

Syntax

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

Parameters

Parameter	Description
url	The path of the OSS object based on which you want to create a raster object. If the OSS object is in Network Common Data Form (NetCDF) and contains a subset, you can specify the subset in the `:<name>` format.
storageOption	A JSON string that describes the chunks for storing the pyramid of the raster object.

The following table describes fields in the storageOption parameter.

Field	Description	Type	Format	Default value	Setting note

Field	Description	Type	Format	Default value	Setting note
chunkdim	The size of each chunk that is used to store the data of the raster object.	string	(w, h, b)	Same as the size of each chunk in the OSS object	None.
interleaving	The interleaving type of the new raster object.	string	None.	bsq	Valid values: • bip: band interleaved by pixel (BIP) • bil: band interleaved by line (BIL) • bsq: band sequential (BSQ) • auto: a value that is specified by the system based on the OSS object

Note You need to change the default values of the chunkdim and interleaving fields only in some cases:

- Users want to view the raster object based on multiband (red, green, and blue) RGB combination, but the value of the interleaving field is bsq. In this case, change the value of the interleaving field to bip.
- The chunks of some images that are used to render the raster object contain 1 row and n columns in size. However, users request chunks that contain 256 rows and 256 columns in size. In this case, you must change the value of the chunkdim field to the requested chunk size.

Description

You can specify the path of the OSS object in the following format: `oss://access_id:secret_key@endpoint/path_to/file`. The endpoint is optional. If you do not specify the endpoint, the system automatically finds the endpoint and you must make sure that the path starts with a forward slash (/).

The endpoint is the domain name that you can use to access the OSS bucket where the OSS object is stored. To maximize data import performance, make sure that the ApsaraDB RDS for PostgreSQL instance resides in the same region as the OSS bucket. For more information, see OSS Endpoint.

Examples

```
-- Specify the AccessKey ID, AccessKey secret, and endpoint in the URL of an OSS object to create a raster obj
ect.
Select ST_CreateRast('OSS://<ak>:<ak_secret>@oss-cn-beijing-internal.aliyuncs.com/mybucket/data/image
.tif');
-- Specify the chunk size and the interleaving type of a raster object.
Select ST_CreateRast('OSS://<ak>:<ak_secret>@oss-cn-beijing-internal.aliyuncs.com/mybucket/data/image
.tif', '{"chunkdim":"(256,256,3)","interleaving":"auto"}');
-- Specify an OSS object that is in NetCDF and contains a subset.
Select ST_CreateRast('OSS://<ak>:<ak_secret>@oss-cn-beijing-internal.aliyuncs.com/mybucket/data/image
.nc:hcc');
```

4.4. Import and export

4.4.1. ST_ImportFrom

This topic describes the ST_ImportFrom function, which is used to import an Object Storage Service (OSS) object into a raster object in PolarDB.

Syntax

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

Parameters

Parameter	Description
chunkTableName	The name of the OSS object. The OSS object is a chunk table. The name must comply with the table naming conventions of PolarDB.
url	The path of the OSS object. For more information, see the description of the url parameter in ST_CreateRast.
storageOption	The JSON string that is used to specify the storage information of the raster object.
importOption	The JSON string that is used to specify the import-related parameters. PolarDB supports the parallel parameter. This parameter specifies the degree of parallelism. The data type of this parameter is INTEGER. The value of this parameter ranges from 1 to 64. If you do not specify the parallel parameter, the value of the GUC parameter ganos.parallel.degree is used as the default degree of parallelism. For more information, see ganos.parallel.degree.

The following table describes the parameters in storageOption.

Parameter	Type	Default value	Description
chunking	boolean	true	Specifies whether to store the data as chunks.

Parameter	Type	Default value	Description
chunkdim	string	Same as the value for the original data	The dimensions that are used to chunk data. Format: w, h, b. **Note** This parameter takes effect only when you set the chunking parameter to **true**.
compression	string	lz4	The format that is used to compress the data of the OSS object. Valid values: • none • jpeg • zlib • png • lzo • lz4 • snappy • zstd • jp2k
quality	integer	75	The quality of compression. This parameter is valid only for the JPEG and JPEG 2000 compression formats.
interleaving	string	Same as the value for the original data	The method that is used to interleave the data of the OSS object. Valid values: • bip : Band interleaved by pixel • bil : Band nterleaved by pixel • bsq : Band Sequential
blockendian	string	'NDR'	The sequence that is used to store the bytes in each data chunk. Valid values: • NDR: little endian • XDR: big endian

Parameter	Type	Default value	Description
celltype	string	Same as the value for the original data	The pixel type of the raster object. Valid values: • 1bb: 1 bit • 2bui: 2-bit unsigned integer • 4bui: 4-bit unsigned integer • 8bs: 8-bit signed integer • 8bui: 8-bit unsigned integer • 16bsi: 16-bit signed integer • 16bui: 16-bit unsigned integer • 32bsi: 32-bit signed integer • 32bui: 32-bit unsigned integer • 32bf: 32-bit float • 64bsi: 64-bit signed integer • 64bui: 64-bit unsigned integer • 64bf: 64-bit float

Description

This function is used to create a raster object and import an OSS object into the created raster object.

The following table lists the data types that are supported by this function.

Data type	Full name
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
JPEG	JPEG JFIF File Format

Examples

```
-- Specify only the name and path of the OSS object that you want to import.
Select ST_ImportFrom('chunk_table','OSS://<ak>:<ak_secret>@oss-cn-beijing-internal.aliyuncs.com/mybucket/data/image.tif');
-- Specify a NetCDF image that contains subsets.
Select ST_ImportFrom('chunk_table','OSS://<ak>:<ak_secret>@oss-cn-beijing-internal.aliyuncs.com/mybucket/data/image.nc:hcc');
-- Specify the size per chunk and the compression format during the import process.
Select ST_ImportFrom('chunk_table','OSS://<ak>:<ak_secret>@oss-cn-beijing-internal.aliyuncs.com/mybucket/data/image.tif', '{"chunkdim":"(128,128,3)", "compression":"none"}');
-- Specify the degree of parallelism at import.
Select ST_ImportFrom('chunk_table','OSS://<ak>:<ak_secret>@oss-cn-beijing-internal.aliyuncs.com/mybucket/data/image.nc:hcc', '{}', '{"parallel": 4}');
```

4.4.2. ST_ExportTo

This function exports a raster object as an Object Storage Service (OSS) object.

Syntax

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

Parameters

Parameter	Description
source	The raster object to be exported.
format	The format of the exported data, such as GTiff or BMP.
url	The URL of the exported OSS object. For more information, see the description of the url parameter in ST_CreateRast.
level	The pyramid level.

Description

If the raster object is successfully exported, the function returns true. If the raster object fails to be exported, the function returns false.

The format parameter specifies the format of the exported data. The following table lists the common formats.

Format	Full name
BMP	Microsoft Windows Device Independent Bitmap(.bmp)
EHdr	ESRI .hdr Labelled
ENVI	ENVI .hdr Labelled Raster

Format	Full name
GTiff	TIFF/BigTIFF/GeoTIFF(.tif)
HFA	Erdas Imagine .img
RST	Idrisi Raster Format
INGR	Intergraph Raster Format

Example

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

4.4.3. ST_AsImage

This topic describes the ST_AsImage function, which converts a raster into a BYTEA-type image.

Syntax

```
bytea ST_AsImage(raster raster_obj,
    box extent,
    integer pyramidLevel default 0,
    cstring bands default '',
    cstring format default 'PNG',
    cstring option default '');
```

Parameters

Parameter	Description
raster_obj	The raster that you want to convert.
extent	The extent of the image. The geographic coordinate system is used by default.
pyramidLevel	The pyramid layer at which the image resides. Valid values start from 0. Default value: 0.
bands	The list of bands based on which you want to obtain the image. Valid values start from 0. Examples: `'0-2'` and `'1,2,3'`. This parameter is empty by default. If the image is in the JPEG format, set this parameter to 1 or 3. If the image is in the PNG format, set this parameter to 1, 2, 3, or 4. The first three bands are used by default.
format	The format of the image. Valid values: • PNG • JPEG
option	The options that are used to convert JSON-formatted strings.

The following table describes the fields in the option parameter.

Field	Description	Type	Default value	Setting notes
nodata	Specifies whether to use NoData values.	bool	false	• true: NoData values are processed. • false: NoData values are processed as normal values.
nodataValue	The NoData value of the raster.	integer	0	If the nodata parameter is set to true, you must specify the NoData value.
rast_coord	Specifies whether the input box is specified by using pixel coordinates.	bool	false	If pixel coordinates are used, x indicates the column No. of the pixel and y indicates the row No. of the pixel. Both the column No. and the row No. start from 0.
strength	Specifies whether to implement enhancement in the display.	string	none	Valid values: • none: Enhancement is not implemented. • stats: Enhancement is implemented by using stretching based on statistical values.
quality	The quality of compression.	integer	75	Valid values: 1 to 100.

Description

- This function returns a BYTEA-type image.
- Up to 100 MB of cropped data can be cached by default, which indicates that up to 100 MB of data can be returned. To adjust the limit on the image size, you can specify the cache size by using the **ganos.raster.clip_max_buffer_size** parameter.
- The following values are valid for this parameter:
  - 1: The raster has a single band based on which the raster can be converted into a grayscale image.
  - 2: The raster has a single band, based on which the raster can be converted into a grayscale image, and the Alpha band.
  - 3: The raster has three bands, which are the R band, G band, and B band.
  - 4: The raster has four bands, which are the R band, G band, B band, and Alpha band.

Examples

```
--Specify the size of cropped data that can be cached.
SELECT ST_AsImage(raster_obj,
          '(-180,-90), (0,0)'::Box)
FROM raster_table
WHERE id=1;
--Specify the pyramid layer at which the image resides.
SELECT ST_AsImage(raster_obj,
          '(-180,-90), (0,0)'::Box,
          1)
FROM raster_table
WHERE id=1;
--Specify the cropping range of a band.
SELECT ST_AsImage(raster_obj,
          '(-180,-90), (0,0)'::Box,
          1,
          '0-2')
FROM raster_table
WHERE id=1;
--Specify the compression format.
SELECT ST_AsImage(raster_obj,
          '(-180,-90), (0,0)'::Box,
          1,
          '0-2',
          'PNG')
FROM raster_table
WHERE id=1;
--Specify to implement enhancement by using stretching based on statistical values.
SELECT ST_AsImage(rast,
          '(-180,-90), (0,0)'::Box,
          0,
          '',
          'PNG',
          '{"nodata":"false", "nodatavalue":"0","rast_coord":"false", "strength":"stats", "quality":"75"}')
FROM raster_table
WHERE id=1;
--Specify the pixel coordinates based on which you want to crop the image and implement enhancement by using stretching based statistical values.
SELECT ST_AsImage(rast,
          '(0,0), (200,100)'::Box,
          0,
          '',
          'PNG',
          '{"nodata":"false", "nodatavalue":"0","rast_coord":"true", "strength":"stats", "quality":"75"}')
FROM raster_table
WHERE id=1;
```

4.4.4. ST_AsPNG

This topic describes the ST_AsPNG function, which converts a raster into a BYTEA-type PNG image.

Syntax

```
bytea ST_AsPNG(raster raster_obj,box extent,
        integer pyramidLevel default 0,
        cstring bands default '',
        cstring option default '');
```

Parameters

Parameter	Description
raster_obj	The raster that you want to convert.
extent	The extent of the image. The geographic coordinate system is used by default.
pyramidLevel	The pyramid layer at which the image resides. Valid values start from 0. Default value: 0.
bands	The list of bands based on which you want to obtain the image. Valid values start from 0. Examples: `'0-2'` and `'1,2,3'`. This parameter is empty by default. The image is in the PNG format. Therefore, set this parameter to 1, 2, 3, or 4. The first three bands are used by default.
option	The options that are used to convert JSON-formatted strings.

The following table describes the fields in the option parameter.

Field	Description	Type	Default value	Setting notes
nodata	Specifies whether to use NoData values.	bool	false	• true: NoData values are processed. • false: NoData values are processed as normal values.
nodataValue	The NoData value of the raster.	integer	0	If the nodata parameter is set to true, you must specify the NoData value.
rast_coord	Specifies whether the input box is specified by using pixel coordinates.	bool	false	If pixel coordinates are used, x indicates the column number of the pixel and y indicates the row number of the pixel. Both the column number and the row number start from 0.
strength	Specifies whether to implement enhancement in the display.	string	none	Valid values: • none: Enhancement is not implemented. • stats: Enhancement is implemented by using stretching based on statistical values.

Field	Description	Type	Default value	Setting notes
quality	The quality of compression.	integer	75	Valid values: 1 to 100.

Description

- This function returns a BYTEA-type PNG image.
- Up to 100 MB of cropped data can be cached by default, which indicates that up to 100 MB of data can be returned. To adjust the limit on the image size, you can specify the cache size by using the **ganos.raster.clip_max_buffer_size** parameter.
- The following values are valid for this parameter:
  - 1: The raster has a single band based on which the raster can be converted into a grayscale image.
  - 2: The raster has a single band, based on which the raster can be converted into a grayscale image, and the Alpha band.
  - 3: The raster has three bands, which are the R band, G band, and B band.
  - 4: The raster has four bands, which are the R band, G band, B band, and Alpha band.

Examples

```
--Specify the size of cropped data that can be cached.
SELECT ST_AsPNG(raster_obj,
          '(-180,-90), (0,0)'::Box)
FROM raster_table
WHERE id =1;
--Specify the pyramid layer at which the image resides.
SELECT ST_AsPNG(raster_obj,
          '(-180,-90), (0,0)'::Box,
          1)
FROM raster_table
WHERE id =1;
--Specify the cropping range of a band.
SELECT ST_AsPNG(raster_obj,
          '(-180,-90), (0,0)'::Box,
          1,
          '0-2')
FROM raster_table
WHERE id =1;
--Specify to implement enhancement by using stretching based on statistical values.
SELECT ST_AsPNG(rast,
          '(-180,-90), (0,0)'::Box,
          0,
          '',
          '{"nodata":"false", "nodatavalue":"0","rast_coord":"false", "strength":"stats", "quality":"75"}')
FROM raster_table
WHERE id =1;
```

4.4.5. ST_AsJPEG

This topic describes the ST_AsJPEG function, which converts a raster into a BYTEA-type JPEG image.

Syntax

```
bytea ST_AsJPEG(raster raster_obj,
        box extent,
        integer pyramidLevel default 0,
        cstring bands default '',
        cstring option default '');
```

Parameters

Parameter	Description
raster_obj	The raster that you want to convert.
extent	The extent of the image. The geographic coordinate system is used by default.
pyramidLevel	The pyramid layer at which the image resides. Valid values start from 0. Default value: 0.
bands	The list of bands based on which you want to convert the raster. Valid values start from 0. Examples: `'0-2'` and `'1,2,3'`. This parameter is empty by default. The image is in the JPEG format. Therefore, set this parameter to 1, 2, 3, or 4. The first three bands are used by default.
option	The options that are used to convert JSON-formatted strings.

The following table describes the fields in the option parameter.

Field	Description	Type	Default value	Setting notes
nodata	Specifies whether to use NoData values.	bool	false	• true: NoData values are processed. • false: NoData values are processed as normal values.
nodataValue	The NoData value of the raster.	integer	0	If the nodata parameter is set to true, you must specify the NoData value.
rast_coord	Specifies whether the input box is specified by using pixel coordinates.	bool	false	If pixel coordinates are used, x indicates the column No. of the pixel and y indicates the row No. of the pixel. Both the column No. and the row No. start from 0.

Field	Description	Type	Default value	Setting notes
strength	Specifies whether to implement enhancement in the display.	string	none	Valid values: • none: Enhancement is not implemented. • stats: Enhancement is implemented by using stretching based on statistical values.
quality	The quality of compression.	integer	75	Valid values: 1 to 100.

Description

- This function returns a BYTEA-type image.
- Up to 100 MB of cropped data can be cached by default, which indicates that up to 100 MB of data can be returned. To adjust the limit on the image size, you can specify the cache size by using the **ganos.raster.clip_max_buffer_size** parameter.
- The following values are valid for this parameter:
  - 1: The raster has a single band based on which the raster can be converted into a grayscale image.
  - 3: The raster has three bands, which are the R band, G band, and B band.

Examples

```
--Specify the size of cropped data that can be cached.
SELECT ST_AsJPEG(raster_obj,
          '(-180,-90), (0,0)'::Box)
FROM raster_table
WHERE id =1;
--Specify the pyramid layer at which the image resides.
SELECT ST_AsJPEG(raster_obj,
          '(-180,-90), (0,0)'::Box,
          1)
FROM raster_table
WHERE id =1;
--Specify the cropping range of a band.
SELECT ST_AsJPEG(raster_obj,
          '(-180,-90), (0,0)'::Box,
          1,
          '0-2')
FROM raster_table
WHERE id =1;
--Specify to implement enhancement by using stretching based on statistical values.
SELECT ST_AsJPEG(rast,
          '(-180,-90), (0,0)'::Box,
          0,
          '',
          '{"nodata":"false", "nodatavalue":"0","rast_coord":"false", "strength":"stats", "quality":"75"}')
FROM raster_table
WHERE id =1;
```

4.4.6. ST_AsDatasetFile

This topic describes the ST_AsDatasetFile function. This function is used to convert a specific part of a raster object into a BYTEA-type file.

Syntax

```
setof record ST_AsDatasetFile(raster raster_obj,
                box extent,
                integer pyramidLevel default 0,
                cstring bands default '',
                cstring format default 'GTiff',
                cstring create_option default '{}',
                cstring process_option default '{}',
                out ext cstring,
                out data bytea);
```

Parameters

Parameter	Description
raster_obj	The raster object whose part you want to convert.
extent	The part of data that you want to extract. By default, the part of data is specified based on a geographic coordinate system (GCS).
pyramidLevel	The pyramid level from which you want to extract data. Valid values start from 0. Default value: 0.
bands	The IDs of the bands from which you want to extract data. Valid band IDs start from 0. The value of this parameter can be a range or array of band IDs. Examples: `'0-2'` and `'1,2,3'`. Default value: `''`. The default value specifies that Ganos extracts data from all bands.
format	The format of the file. For more information, see ST_RasterDrivers.
create_option	The options based on which you want to create a dataset by using the extracted data. The value of this parameter is a JSON string. For more information, see ST_RasterDrivers.
process_option	The options based on which you want to process the extracted data. The value of this parameter is a JSON string. The rast_coord option specifies whether pixel coordinates are used to specify the input bounding box. If pixel coordinates are used to specify the input bounding box, the x-coordinate specifies the column No. of a pixel and the y-coordinate specifies the row No. of a pixel. The valid values of the x-coordinate and y-coordinate start from 0.
ext	The format of the file. The supported file formats include TIF and XML.
data	The BYTEA-type data of the file.

Description

- The file format that you specify must be supported by a driver in Ganos. Therefore, you must set the can_asfile parameter in the ST_RasterDrivers function to true. For more information, see ST_RasterDrivers.
- The default size of the cache that stores the extracted data is 100 MB. This means that up to 100 MB of extracted data can be returned. If you want to adjust the size of the extracted data that is returned, you can change the cache size by using the ganos.raster.clip_max_buffer_size parameter.
- The value of the create_option parameter can be obtained from the create_options parameter of the ST_RasterDrivers function. For more information, see ST_RasterDrivers.

Examples

```
--Specify the part of data that you want to extract.
SELECT ST_AsDatasetFile(raster_obj,
          '(-180,-90), (0,0)'::Box)
FROM raster_table
WHERE id =1;
--Specify the pyramid level from which you want to extract data.
SELECT ST_AsDatasetFile(raster_obj,
          '(-180,-90), (0,0)'::Box,
          1)
FROM raster_table
WHERE id =1;
--Specify the IDs of the bands from which you want to extract data.
SELECT ST_AsDatasetFile(raster_obj,
          '(-180,-90), (0,0)'::Box,
          1,
          '0-2')
FROM raster_table
WHERE id =1;
--Specify that Ganos saves the extracted data as a GIF file.
SELECT ST_AsDatasetFile(raster_obj,
          '(-180,-90), (0,0)'::Box,
          1,
          '0-2',
             'GIF')
FROM raster_table
WHERE id =1;
--Specify the file format and options that are used to process the extracted data.
SELECT ST_AsDatasetFile(raster_obj,
          '(-180,-90), (0,0)'::Box,
          1,
          '0-2',
             'GTiff',
             '{"blockxsize":256, "blockysize":256, "compress": "DEFLATE"}')
FROM raster_table
WHERE id =1;
--Specify that pixel coordinates are used to specify the input bounding box.
SELECT ST_AsDatasetFile(raster_obj,
          '(0,0), (100,100)'::Box,
          1,
          '0-2',
             'GTiff',
             '{}',
             '{"rast_coord":"true"}')
FROM raster_table
WHERE id =1;
```

4.5. Pyramid operations

4.5.1. ST_BuildPyramid

This topic describes the ST_BuildPyramid function, which is used to create a pyramid for a raster object.

Syntax

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

Parameters

Parameter	Description
source	The name of the raster object.
chunkTableName	The name of the chunk table that is stored in the pyramid. This parameter takes effect only when the raster object is stored in an Object Storage Service (OSS) bucket.
pyramidLevel	The number of levels in the pyramid. The value -1 specifies that the largest number of levels are created in the pyramid.
algorithm	The resampling algorithm that is used to create the pyramid. Valid values: • Near • Average • Bilinear • Cubic
storageOption	The JSON string that is used to specify the storage-related parameters. These parameters are used to describe the chunk storage information of the pyramid. This JSON string takes effect only when the raster object is stored in an OSS bucket.
buildOption	The JSON string that is used to specify the pyramid building-related parameters. PolarDB supports the parallel parameter. This parameter specifies the degree of parallelism. The data type of this parameter is INTEGER. The value of this parameter ranges from 1 to 64. If you do not specify the parallel parameter, the value of the GUC parameter ganos.parallel.degree is used as the default degree of parallelism. For more information, see ganos.parallel.degree. **Note** If you run threads in parallel to create the pyramid, transactions are not supported. If the pyramid cannot be created or you need to roll transactions back, you can invoke the ST_deletePyramid function to delete the pyramid. For more information, see ST_deletePyramid.

The following table describes the parameters in storageOption.

Parameter	Type	Description

Parameter	Type	Description
chunkdim	string	The dimensions that are used to chunk the data of the raster object. The value of this parameter follows the `(w, h, b)` format. By default, the size per chunk is obtained from the original image data.
interleaving	string	The method that is used to interleave the data of the raster object. • bip: band interleaved by pixel (BIP) • bil: band interleaved by line (BIL) • bsq: band sequential (BSQ) (This is the default value.) • auto: an interleaving method that is specified by this function
compression	string	The format into which you want to compress the data of the raster object. Valid values: • none • jpeg • zlib • png • lzo • LZ4 (This is the default value.) • zstd • snappy • jp2k
quality	integer	The quality of compression. This parameter takes effect only when you use the JPEG or JPEG 2000 compression format. Default value: 75.

Description

This function supports GPU-accelerated computing. By default, in an active environment with GPUs, Ganos automatically enables GPU-accelerated computing.

Examples

```
Update raster_table set raster_obj = ST_BuildPyramid(raster_obj) where id = 1;
Update raster_table set raster_obj = ST_BuildPyramid(raster_obj, 'chunk_table') where id = 2;
-- Use JPEG 2000 to compress the data of the raster object.
-- Make sure that all bands are within one chunk.
Update raster_table set raster_obj = ST_BuildPyramid(
  raster_obj,
  -1,
  'Near',
  'chunk_table',
  '{"compression":"jp2k", "quality": 75, "chunkdim":"(256,256,4)","interleaving":"auto"}')
where id = 3;
-- Create the pyramid by using parallel operations.
Update raster_table set raster_obj = ST_BuildPyramid(
  raster_obj,
  -1,
  'Near',
  'chunk_table',
  '{"compression":"jp2k", "quality": 75, "chunkdim":"(256,256,4)","interleaving":"auto"}',
  '{"parallel":4}')
where id = 3;
```

4.5.2. ST_deletePyramid

This function deletes the pyramid for a raster object.

Syntax

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

Parameters

Parameter	Description
source	The raster object.

Description

This function deletes the pyramid, resets the metadata, and deletes the chunk data for the raster object.

Example

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

4.5.3. ST_BestPyramidLevel

This function computes the best pyramid level based on the world space, width, and height of a viewport.

Syntax

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

Parameters

Parameter	Description
rast	The raster object.
box	The world space of the viewport, in the format of `((minX,minY),(maxX,maxY))`.
width	The width of the viewport, in pixels.
height	The height of the viewport, in pixels.

Description

The raster object must have a valid spatial reference system identifier (SRID).

Examples

```
Select ST_BestPyramidLevel(raster_obj, '((128.0, 30.0),(128.5, 30.5))', 800, 600) from raster_table where id = 10;
----------------------
3
```

4.6. Coordinate system conversion

4.6.1. ST_Rast2WorldCoord

This function computes the world coordinates of a cell by using an affine transformation formula based on the pixel coordinates and pyramid level of the cell.

Syntax

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

Parameters

Parameter	Description
raster_obj	The raster object.
pyramidLevel	The pyramid level.
row	The row number of the cell.

Parameter	Description
column	The column number of the cell.
geom	The geometry needs to be converted. The x coordinate of the geometry represents the column number in the raster and the y coordinate of the geometry represents the row number .

Description

The raster object must have a valid spatial reference system identifier (SRID).

Examples

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

4.6.2. ST_World2RastCoord

This function computes the pixel coordinates of a cell by using an inverse affine transformation formula based on the world coordinates and pyramid level of the cell.

Syntax

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

Parameters

Parameter	Description
raster_obj	The raster object.
pyramidLevel	The pyramid level.
coord	The world coordinates of the cell.
geom	The geometry to be converted.

Description

The raster object must have a valid spatial reference system identifier (SRID).

A geometry will be returned. The x coordinate is the column number in raster and the y coordinate is the row number.

Example

```
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)'::geometry)))
FROM tb_world2rast;
 st_astext
--------------
 POINT(511 0)
```

4.7. Pixel value operations

4.7.1. ST_AddZ

This topic describes the ST_AddZ function, which specifies the Z coordinate of a geometry based on the band of the raster converted from the geometry.

Syntax

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

Parameters

Parameter	Description
source	The raster from which the geometry is converted.
geom	The geometry whose Z coordinate you want to specify.
pyramid	The pyramid level at which the raster resides. Valid values start from 0. Default value: 0.
band	The band of the raster. Valid values start from 0. Default value: 0.

Description

This function specifies the Z coordinate of a geometry based on the band of the raster converted from the geometry. If the raster is georeferenced, this function returns a geographic coordinate of the geometry. Otherwise, this function returns a pixel coordinate of the geometry.

> **Note** All of the points on the geometry must fall within the raster.

Example

```
SELECT ST_AddZ(rast_object, ST_GeomFromText('POINT(120.5 30.6)', 4326), 0, 0)
FROM raster_table;
------------------------
POINT Z(120.5 30.6 27)
SELECT ST_AddZ(rast_object, ST_GeomFromText('POINT(120.5 30.6)', 4326), 1, 1)
FROM raster_table;
------------------------
POINT Z(120.5 30.6 115)
```

4.7.2. ST_ClipDimension

This function computes the raster space of a clipped area.

Syntax

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

Parameters

Parameter	Description
raster_obj	The raster object.
pyramidLevel	The pyramid level.
box	The world space of the clipped area.

Description

The raster object must have a valid spatial reference system identifier (SRID).

Examples

```
Select ST_ClipDimension(raster_obj, 2, '((128.0, 30.0),(128.5, 30.5))') from raster_table where id = 10;
------------------------------------
'((200, 300),(600, 720))'
```

4.7.3. ST_Clip

This function clips a raster object.

Syntax

```
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)
```

Parameters

Parameter	Description
raster_obj	The raster object.
pyramidLevel	The pyramid level.
extent	The area to be clipped, in the format of `'((minX,minY),(maxX,maxY))'`.
boxType	The coordinate type of the area to be clipped. Valid values: • Raster: pixel coordinates • World: world coordinates
destSrid	The spatial reference system identifier (SRID) of the output cell subset.
geometry	The geometry object used for clipping.
bands	The sequence numbers of bands to be clipped, in the format of `'0-2'` or `'1,2,3'`. The sequence number starts from 0. Default value: empty string (`''`). It indicates that all bands are to be clipped.
nodata	The array of NoData values in the format of float8[]. If the number of NoData values is fewer than the number of bands to be clipped, the predefined NoData value of a band is used to fill the area after the band is clipped. If a band has no predefined NoData value, the value 0 is used to fill the area after the band is clipped.
clipOption	The clipping options. The value is a JSON-formatted string.
storageOption	The storage options for the output. The value is a JSON-formatted string.

The following table describes the clipOption parameters.

Parameter	Type	Default value	Description

Parameter	Type	Default value	Description
window_clip	BOOLEAN	false	Specifies whether to use the bounding box of the geometry object to clip the raster object. Valid values: • true: uses the minimum bounding rectangle (MBR) of the geometry object. • false: uses the geometry object.
rast_coord	BOOLEAN	false	Specifies whether the geometry object to clip is in raster coordinates. If true, the x coordinate of the geometry represents the column number and the y coordinate represents the row number.

The following table describes fields in the storageOption parameters.

Parameter	Type	Default value	Description
compression	STRING	lz4	The type of the compression algorithm. Valid values: • none • jpeg • zlib • png • lzo • lz4
quality	INTEGER	75	The compression quality. This parameter takes effect only when the value of the compression parameter is set to jpeg.
interleaving	STRING	Same as the original raster object	The interleaving type. Valid values: • bip: band interleaved by pixel (BIP) • bil: band interleaved by line (BIL) • bsq: band sequential (BSQ)
endian	STRING	Same as the original raster object	The endian format. Valid values: • NDR: little endian format • XDR: big endian format

Description

The default clipping cache is 100 MB, which indicates that only 100 MB of data can be returned. To adjust the limit on the output size, you can use the ganos.raster.clip_max_buffer_size parameter to set the size of the cache.

Examples

```
Select ST_Clip(raster_obj, 0, '((128.980,30.0),(129.0,30.2))', 'World');
Select ST_Clip(raster_obj, 0, '((128.980,30.0),(129.0,30.2))', 'World', 4326);
-- Use a geometry object to clip a raster object.
-- Use the default clipping settings.
SELECT (ST_CLIP(rast, ST_geomfromtext('Polygon((0 0, 45 45, 90 45, 45 0, 0 0))', 4326), 0)).* from clip_table w
here id =1
-- Use white as the background color and compress the output into a PNG image.
SELECT (ST_CLIP(rast, ST_geomfromtext('Polygon((0 0, 45 45, 90 45, 45 0, 0 0))', 4326), 0, '', ARRAY[254,254,25
4], '', '{"compression":"png","interleaving":"bip"}')).* from clip_table where id =1;
-- Use the window of a geometry object to clip a raster object.
SELECT (ST_CLIP(rast, ST_geomfromtext('Polygon((0 0, 45 45, 90 45, 45 0, 0 0))', 4326), 0, '', NULL, '{"window_
clip":true}', '')).* from clip_table where id =1;
```

4.7.4. ST_ClipToRast

This function clips a raster object by using a specified geometry object and returns the clipping result as a new raster object.

Syntax

```
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 '')
```

Parameters

Parameter	Description
raster_obj	The raster object to be clipped.
pyramidLevel	The pyramid level.
geometry	The geometry object that is used for clipping.
bands	The bands to be clipped, in the format of `'0-2'` or `'1,2,3'`. It starts from 0. Default value: `''`. The default value indicates that all the bands are clipped.
nodata	The nodata values in the format of float8[]. If the number of the values is less than the number of bands, the nodata values that are specified for the bands are filled. If no nodata value is specified for the bands, the value 0 is filled.
clipOption	The clipping option that is represented by a JSON string.
storageOption	The storage option that is represented by a JSON string in the returned result.

The following table describes the parameters of clipOption.

Parameter	Type	Default value	Description
window_clip	bool	false	Specifies whether to use the bounding box of a geometry for clipping. Valid values: • true: uses the minimum bounding rectangle (MBR) of the geometry for clipping. • false: uses the geometry object for clipping.
rast_coord	bool	false	Specifies whether the passed geometry uses pixel coordinates. If the pixel coordinates are used, the x coordinate represents the column number of the pixel and the y coordinate represents the row number of the pixel.

The following table describes the parameters of storageOption.

Parameter	Type	Default value	Description
chunking	boolean	Same as the original raster	Specifies whether to store data as chunks.
chunkdim	string	Same as the original raster	The dimension information about the chunk. This parameter only takes effect when the chunking parameter is set to true.
chunktable	string	''	The name of the chunk table. If the '' value is passed, a temporary chunk table that has a random name is generated to store data. This temporary table is valid in only the current session. If you need to retain an accessible clipping object, you must specify the name of the chunk table.
compression	string	lz4	The type of the compression algorithm. Valid values: • none • jpeg • zlib • png • lzo • lz4
quality	integer	75	The compression quality. This parameter takes effect for only the jpeg compression algorithm.
interleaving	string	Same as the original raster	The interleaving method. Valid values: • bip: band interleaved by pixel (BIP) • bil: band interleaved by line (BIL) • bsq: band sequential (BSQ)

Parameter	Type	Default value	Description
endian	string	Same as the original raster	The endian. Valid values: • NDR: little endian • XDR: big endian

Description

- If NULL or '' is passed for the chunktable parameter, a temporary chunk table that has a random name is generated to store data. This temporary table is valid in only the current session. If you need to retain an accessible clipping object, you must specify the name of the chunk table.
- The default cache size for clipping is 100 MB. This indicates that up to 100 MB of clipping result data can be returned. If you need to adjust the size of the returned result, you can specify the cache size by using the ganos.raster.clip_max_buffer_size parameter.

Examples

```
-- Create a permanent table.
CREATE TEMP TABLE rast_clip_result(id integer, rast raster);
-- Create a temporary table.
CREATE TEMP TABLE rast_clip_result_temp(id integer, rast raster);
-- Use the default clipping settings and store the result in a temporary table.
INSERT INTO rast_clip_result_temp(id, rast)
select 1, ST_ClipToRast(rast, ST_geomfromtext('Polygon((0 0, 45 45, 90 45, 45 0, 0 0))', 4326), 0)
from clip_table
where id =1;
-- Use white as the background color for padding and store the result in a permanent table.
INSERT INTO rast_clip_result(id, rast)
select 2, ST_ClipToRast(rast, ST_geomfromtext('Polygon((0 0, 45 45, 90 45, 45 0, 0 0))', 4326), 0, '', ARRAY[254, 254,254], '', '{"chunktable":"clip_rbt"}')
from clip_table
where id =1;
-- Use the window of a geometry for clipping.
INSERT INTO rast_clip_result_temp(id, rast)
SELECT 3, ST_ClipToRast(rast, ST_geomfromtext('Polygon((0 0, 45 45, 90 45, 45 0, 0 0))', 4326), 0, '', NULL, '{"
window_clip":true}', '')
from clip_table
where id =1;
```

4.7.5. ST_Values

This function returns the geographic coordinates and values of all the cells in a raster object that intersect with a geometry object.

Syntax

```
set of record ST_Values(raster raster_obj,
        geometry geom,
        integer pyramidLevel default 0,
        cstring bands default '', /* All bands */
        cstring clipOption default '',
        out point coords,
        out integer band,
        out float8 value)
```

Parameters

Parameter	Description
raster_obj	The raster object.
pyramidLevel	The pyramid level.
geometry	The geometry object used for clipping.
bands	The sequence numbers of bands to be clipped, in the format of `'0-2'` or `'1,2,3'`. The sequence number starts from 0. Default value: null string `''`. It indicates that all bands are to be clipped.
clipOption	The clipping options. The value is a JSON-formatted string.
coords	The output field that indicates the geographic coordinates of a pixel value.
band	The output field that indicates the band sequence number of a pixel value.
value	The output field that indicates a pixel value.

The following table describes the clipOption parameter.

Parameter	Type	Default value	Description
window_clip	Boolean	false	Specifies whether to use the bounding box of the geometry object to clip the raster object. Valid values: • true: uses the minimum bounding rectangle (MBR) of the geometry object. • false: uses the geometry object.

Description

- Both the geometry object and the raster object must have a valid spatial reference system identifier (SRID). Their SRIDs must be consistent.
- The default clipping cache is 100 MB, which indicates that only 100 MB of data can be returned. To adjust the limit on the output size, you can use the ganos.raster.clip_max_buffer_size parameter to set the size of the cache.

Examples

```
-- Use a geometry object of the point type to clip the raster object.
SELECT ( ST_Values(rast, ST_geomfromtext('POINT(128.135 29.774)', 4326)::geometry, 0,'{"window_clip":"true"}')).*
from rat_clip WHERE id=1;
   coord    | band | value
--------------+------+-------
 (127.8,29.7) |   0 |   11
 (127.8,29.7) |   1 |   10
 (127.8,29.7) |   2 |   50
(3 rows)
-- Use a geometry object of the linestring type to clip the raster object.
SELECT ( ST_Values(rast, ST_geomfromtext('LINESTRING(0 0, 45 45, 90 0, 135 45)', 4326)::geometry, 0)).*
from rat_clip WHERE id=1 limit 10;
   coord    | band | value
-------------+------+-------
 (44.1,45)  |   0 |  115
 (44.1,45)  |   1 |  112
 (44.1,45)  |   2 |   69
 (45,45)    |   0 |  122
 (45,45)    |   1 |  117
 (45,45)    |   2 |   75
 (134.1,45) |   0 |   37
 (134.1,45) |   1 |   64
 (134.1,45) |   2 |   13
 (43.2,44.1) |   0 |   66
(10 rows)
```

4.7.6. ST_Update

This function uses a source raster object to update a destination raster object.

Syntax

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

Parameters

Parameter	Description
source	The source raster object.
dest	The destination raster object.

Examples

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

4.7.7. ST_MosaicFrom

This topic describes the ST_MosaicFrom function, which performs a mosaic operation to combine multiple raster objects into a new raster object.

Syntax

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

Parameters

Parameter	Description
source	The name of the raster objects that you want to combine.
chunkTableName	The name of the chunk table for storing the new raster object. The name must comply with the table naming conventions of ApsaraDB RDS for PostgreSQL.

Description

This function combines multiple raster objects into a new one.

All of the raster objects that you want to combine must meet the following requirements:

- They have the same number of bands.
- They are all geographically referenced, or none of them is geographically referenced. If all of them are geographically referenced, world coordinates (geographic coordinates) are used for the mosaic operation.
- Although they can have different pixel types, they must have the same spatial reference system identifier (SRID) and affine parameters if world coordinates are used for the mosaic operation.

You must configure the following parameter.

Parameter	Type	Description
ganos.raster.mosaic_must_same_nodata	boolean	Specifies whether the values of NoData in a data source must be the same during the mosaic operation. Valid values: true and false. The values of NoData are not changed during the mosaic operation. If you set this parameter to false, the semantics of the pixels after the mosaic operation may be changed. Example: `Set ganos.raster.mosaic_must_same_nodata = false;`

Example

```
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_table WHERE id < 10), 'chunk_table_mosaic') WHERE id = 11;
```

4.7.8. ST_MosaicTo

This topic describes the ST_MosaicTo function, which performs a mosaic operation to integrate a raster object or object set into another raster object.

Syntax

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

Parameters

Parameter	Description
raster_obj	The raster object into which you want to integrate another raster object or object set.
source	The raster object or object set that you want to integrate it into another raster object or object set.

Description

The source and destination raster objects must meet the following requirements:

- They have the same number of bands.
- They are all geographically referenced, or none of them is geographically referenced. If all of them are geographically referenced, world coordinates (geographic coordinates) are used for the mosaic operation.
- Although they can have different pixel types, they must have the same spatial reference system identifier (SRID) and affine parameters if world coordinates are used for the mosaic operation.

You must configure the following parameter.

Parameter	Type	Description
ganos.raster.mosaic_must_same_nodata	boolean	Specifies whether the values of NoData in a data source must be the same during the mosaic operation. Valid values: true and false. The values of NoData are not changed during the mosaic operation. If you set this parameter to false, the semantics of the pixels after the mosaic operation may be changed. Example: `Set ganos.raster.mosaic_must_same_nodata = false;`

Example

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

4.8. Overview operations

4.8.1. ST_BuildOverview

This function creates an overview.

Syntax

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

Parameters

Parameter	Description
tableName	The name of the table.
columnName	The name of the raster object column.
pyramidLevel	The pyramid level.
chunkTableName	The name of the chunk table for storing the raster object created as an overview.

This function creates a table-based raster overview.

All the raster objects involved must meet the following requirements:

- They have the same number of bands.
- Either all of them are geographically referenced, or none of them is geographically referenced. If all of them are geographically referenced, world coordinates (geographic coordinates) are used for the mosaic operation.
- Their pixel types can be different. If world coordinates are used for the mosaic operation, they must have the same spatial reference system identifier (SRID) and pixel resolution.

Examples

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

4.8.2. ST_UpdateOverview

This function uses a raster object or object set to update an overview.

Syntax

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

Parameters

Parameter	Description
raster_obj	The destination raster object.

Parameter	Description
source	The source raster object or object set.

Description

All the specified raster objects must meet the following requirements:

- They have the same number of bands.
- Either all of them are geographically referenced, or none of them is geographically referenced. If all of them are geographically referenced, world coordinates (geographic coordinates) are used for the mosaic operation.
- Their pixel types can be different. If world coordinates are used for the mosaic operation, they must have the same spatial reference system identifier (SRID) and pixel resolution.

Examples

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

4.8.3. ST_EraseOverview

This function clears the specified area of a raster object.

Syntax

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

Parameters

Parameter	Description
raster_obj	The raster object.
extent	The area to be cleared, in the format of `'((minX,minY),(maxX,maxY))'`.
type	The coordinate type of the area to be cleared. The value must be Raster or World, where Raster indicates pixel coordinates and World indicates world coordinates.
useNodata	Indicates whether to use the predefined NoData value to fill the area to be cleared. If this parameter is set to false or the NoData value is not defined, 0 is used to fill the area.

Examples

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

4.9. DEM operations

4.9.1. ST_Aspect

This function calculates the aspect from each cell on the surface of a raster object and returns an array of aspects.

Syntax

```
raster ST_Aspect(raster rast, integer pyramid_level, integer band, Box extent, BoxType type, cstring storage Option);
```

Parameters

Parameter	Description
rast	The raster object.
pyramid_level	The pyramid level.
Band	The sequence number of the band.
box	The area that you want to analyze. Specify the value in the `'((minX,minY),(maxX,maxY))'` format.
type	The type of coordinate that is used to identify the area to be analyzed. You can specify only one value. Valid values: • Raster: pixel coordinates • World: world coordinates
storageOption	The storage option of the raster object. For more information, see ST_ClipToRast.

Description

The ST_Aspect function is used to calculate the aspect from the surface of a raster object. The aspect identifies the downslope direction of the maximum rate of change in a value from each cell to neighboring cells. The aspect can be considered the slope direction. The value of each cell in the output raster object indicates the compass direction of the aspect. The aspect is measured clockwise from the north. Unit: degrees. Valid values: 0 to 360. Cells in the input raster object that are flat and do not have a downslope direction are assigned an aspect of -1 in the output raster object.

The value of a cell in the aspect dataset of the output raster object indicates the slope direction of the cell.

Example:

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

4.9.2. ST_Slope

This function calculates the slope from each cell over the surface of a raster object and returns an array of slopes that are expressed in radians.

Syntax

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

Parameters

Parameter	Description
rast	The raster object.
pyramid_level	The pyramid level.
Band	The sequence number of the band.
box	The area that you want to analyze. Specify the value in the `'((minX,minY),(maxX,maxY))'` format.
type	The type of coordinate that is used to identify the area to be analyzed. You can specify only one value. Valid values: • Raster: pixel coordinates • World: world coordinates
zfactor	The conversion factor that is used to adjust the unit of measurement for the elevation units when the elevation units are different from the horizontal coordinates of the input surface. Default value: 1.
storageOption	The storage option of the raster object. For more information,see ST_ClipToRast.

Description

The ST_Slope function calculates the maximum rate of change in a value from each cell to neighboring cells. The maximum change in elevation over the distance between a cell and eight neighboring cells identifies the steepest downhill descent from the cell.

Example:

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

4.9.3. ST_Hillshade

This function calculates the hillshade from each cell over the surface of a raster object and returns an array of hillshades.

Syntax

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

Parameters

Parameter	Description
rast	The raster object.
pyramid_level	The pyramid level.
band	The sequence number of the band.
extent	The area that you want to analyze. Specify the value in the `'((minX,minY),(maxX,maxY))'` format.
type	The type of coordinate that is used to identify the area to be analyzed. You can specify only one value. Valid values: • Raster: pixel coordinates • World: world coordinates
zfactor	The conversion factor that is used to adjust the unit of measurement for the elevation units when the elevation units are different from the horizontal coordinates of the input surface. Default value: 1.
azimuth	The azimuth angle of the sun. The azimuth angle is measured clockwise from the north. Unit: degrees. Valid values: 0 to 360. Default value: 315 (northwest).
altitude	The altitude angle of the sun above the horizon. Unit: degrees. Valid values: 0 to 90. A value of 90 indicates that the sun is directly overhead.
storageOption	The storage option of the raster object. For more information, see ST_ClipToRast.

Description

The ST_Hillshade function obtains the hypothetical illumination of the surface of a raster object by determining the illumination value from each cell of the raster object. This function assigns a position for a hypothetical light source and calculates the illumination value of each cell in relation to neighboring cells. This function can greatly enhance the visualization of a raster surface for analysis or graphical display, especially when transparency is used.

By default, shadow and light are displayed as shades in gray that are associated with integers from 0 to 255. The darkest shade is represented by 0, and the brightest shade is represented by 255.

Example:

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

4.9.4. ST_Overflow

This function computes the water depth in an area based on the geographic coordinates and precipitation of the area.

Syntax

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

Parameters

Parameter	Description
rast	The raster object, which currently must be a digital elevation model (DEM) that has only one band.
box	The world space of the area to be analyzed, which must be consistent with that of the DEM.
value	The precipitation of the area. Unit: cubic meters.

Description

This function computes the water depth in an area based on the geographic coordinates and precipitation of the area. Currently, the raster object must be a DEM that has only one band.

Examples

```
select st_overflow(rast, '((-195516.0,2230052.0),(-194626.0,2230942.0))'::box, 10000::float8) from t_overflow
where id =1;
                                                                                    st_overflow
------------------------------------------------------------------------------------
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1.43427103409668,0,0.434271034096678,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0.434271034096678,0,0.434271034096678,0,0.434271034096678,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0.434271034096678,0.434271034096678,0.434271034096678,0.434271034096678,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.434271034096678,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0.434271034096678,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0.434271034096678,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.43427103409667
8,0.434271034096678,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.434271034096678,0.434271034096
678,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.434271034096678,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.4
34271034096678,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.434271034096678,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1.43427103409668,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0.434271034096678,0,0,0,0,0}
(1 row)
```

4.9.5. ST_Flow_direction

This function computes the flow direction in an area based on the digital elevation model (DEM) data in the area.

Syntax

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

Parameters

Parameter	Description
rast	The raster object, which currently must be a digital elevation model (DEM) that has only one band.
box	The world space of the area to be analyzed.

Description

This function computes the flow direction in an area based on the DEM data in the area by using the D8 flow direction algorithm.

The raster object must be a DEM value that has only one band.

Example

```
select st_flow_direction(rast, '((-202286.94,2232375.16),(-202135.0,2232225))'::box) from t_overflow where id =2;
                    st_flow_direction
--------------------------------------------------------------------------------
{2,2,2,4,4,8,2,2,2,4,4,8,1,1,2,4,8,4,128,128,1,2,4,8,2,2,1,4,4,4,1,1,1,1,4,16}
(1 row)
```

4.10. Attribute query and update

4.10.1. ST_Name

This function returns the name of a raster object. If no name is specified, the function returns null.

Syntax

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

Parameters

Parameter	Description
rast	The raster object.

Examples

```
select ST_Name(rast) from rat where id=1;
_____________________________________
image1
```

4.10.2. ST_SetName

This function sets the name of a raster object.

Syntax

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

Parameters

Parameter	Description
rast	The raster object.
name	The name of the raster object.

Examples

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

4.10.3. ST_MetaData

This function returns the metadata of a raster object in JSON format.

Syntax

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

Parameters

Parameter	Description
raster_obj	The raster object.

Parameter	Description
band	The sequence number of the band, which starts from 0.
key	The name of the metadata item that you want to query. If you set this parameter to `all`, all metadata items are retutrned in JSON format.

Examples

```
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 and swell","
missing_value":"-32767","NETCDF_DI.
M_time":"1043112","NETCDF_VARNAME":"swh","scale_factor":"0.0001488117874873806","units":"m","_Fill
Value":"-32767"}
```

4.10.4. ST_Width

This function returns the width of a raster object. For more information about how to obtain the width of a chunk, see ST_ChuckWidth.

Syntax

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

Parameters

Parameter	Description
raster_obj	The raster object.

Examples

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

4.10.5. ST_Height

This topic describes the ST_Height function, which obtains the height of a raster.

Syntax

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

Parameters

Parameter	Description
raster_obj	The raster whose height you want to obtain.
pyramid	The pyramid level at which the raster resides. Valid values start from 0.

Example

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

4.10.6. ST_NumBands

This function returns the number of bands in a raster object.

Syntax

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

Parameters

Parameter	Description
raster_obj	The raster object.

Examples

```
select ST_NumBands(raster_obj) from raster_table;
-----------------------------------
3
```

4.10.7. ST_Value

This function returns the value of a cell based on the specified band, row number, and column number.

Syntax

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

Parameters

Parameter	Description
rast	The raster object.
band	The sequence number of the band.
colsn	The column number of the cell.
rowsn	The row number of the cell.
exclude_nodata_value	Specifies whether to exclude the NoData value. Default value: true.

Description

This function returns the value of a cell based on the specified band, row number, and column number. The band sequence number starts from 0. The default band sequence number, row number, and column number are all 0.

Examples

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

4.10.8. ST_RasterID

This function returns the universally unique identifier (UUID) of a raster object.

Syntax

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

Parameters

Parameter	Description
raster_obj	The raster object.

Examples

```
select ST_RasterID(raster_obj) from raster_table;
--------------------------------------
4e692ed0-74e2-42a3-a10d-c28d4ae31982
```

4.10.9. ST_CellDepth

This function returns the pixel depth of a raster object. The pixel depth can be 0, 1, 2, 4, 8, 16, 32, or 64, where 0 indicates that the pixel depth is unknown.

Syntax

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

Parameters

Parameter	Description
raster_obj	The raster object.

Examples

```
select ST_CellDepth(raster_obj) from raster_table;
_____________________________________
8
```

4.10.10. ST_CellType

This function returns the pixel type of a raster object. The pixel type can be 8BSI, 8BUI, 16BSI, 16BUI, 32BSI, 32BUI, 32BF, or 64BF.

Syntax

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

Parameters

Parameter	Description
raster_obj	The raster object.

Examples

```
select st_celltype(raster_obj) from raster_table;
_____________________________________
8BUI
```

4.10.11. ST_InterleavingType

This function returns the interleaving type of a raster object. The interleaving type can be BSQ, BIL, or BIP.

Syntax

```
text ST_InterleavingType(raster raster_obj);
```

Parameters

Parameter	Description
raster_obj	The raster object.

Examples

```
select ST_InterleavingType(raster_obj) from raster_table;
_____________________________________
BSQ
```

4.10.12. ST_TopPyramidLevel

This function returns the highest pyramid level of a raster object.

Syntax

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

Parameters

Parameter	Description
raster_obj	The raster object.

Examples

```
select ST_TopPyramidLevel(raster_obj) from raster_table;
_____________________________________
6
```

4.10.13. ST_Extent

This topic describes the ST_Extent function, which obtains the coordinate range of a raster in the following format: '((maxX,maxY),(minX,minY))'.

Syntax

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

Parameters

Parameter	Description
raster_obj	The raster whose coordinate range you want to obtain.
csOption	The type of coordinate to return.
pyramid	The pyramid level at which the raster resides. Valid values start from 0.

Description

This function obtains the coordinate range of a raster based on the value of the csOption parameter:

- Raster: This function returns pixel coordinates.
- World: This function returns world coordinates.
- WorldFirst: This function preferentially returns world coordinates. If the raster is georeferenced, this function returns world coordinates. Otherwise, this function returns pixel coordinates.

Example

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

4.10.14. ST_ConvexHull

This topic describes the ST_ConvexHull function, which obtains the minimum convex geometry that encloses all geometries within a raster.

Syntax

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

Parameters

Parameter	Description
source	The raster whose minimum convex geometry you want to obtain.
pyramid	The pyramid level at which the raster resides. Valid values start from 0. Default value: 0.

Description

This function obtains the minimum convex geometry that encloses all geometries within a raster.

Example

```
SELECT st_astext(st_convexhull(rast_object))
FROM raster_table;
----------------------------------------------------
 POLYGON((-180 90,180 90,180 -90,-180 -90,-180 90))
--Specify the pyramid level at which the raster resides.
SELECT st_astext(st_convexhull(rast_object, 1))
FROM raster_table;
----------------------------------------------------
 POLYGON((-180 90,180 90,180 -90,-180 -90,-180 90))
```

4.10.15. ST_Envelope

This topic describes the ST_Envelope function, which obtains the bounding box of a raster.

Syntax

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

Parameters

Parameter	Description
source	The raster whose bounding box you want to obtain.
pyramid	The pyramid level at which the raster resides. Valid values start from 0. Default value: 0.

Description

This function obtains the bounding box of a raster.

Example

```
SELECT st_astext(st_envelope(rast_object))
FROM raster_table;
----------------------------------------------------
 POLYGON((-180 90,180 90,180 -90,-180 -90,-180 90))
--Specify the pyramid level at which the raster resides.
SELECT st_astext(st_envelope(rast_object, 1))
FROM raster_table;
----------------------------------------------------
 POLYGON((-180 90,180 90,180 -90,-180 -90,-180 90))
```

4.10.16. ST_Srid

This function returns the spatial reference system identifier (SRID) of a raster object. The SRID and its definition are stored in the spatial_ref_sys table.

Syntax

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

Parameters

Parameter	Description
raster_obj	The raster object.

Examples

```
select ST_Srid(raster_obj) from raster_table where id=1;
_____________________________________
4326
```

4.10.17. ST_SetSrid

This function sets the spatial reference system identifier (SRID) of a raster object. The SRID and its definition are stored in the spatial_ref_sys table.

Syntax

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

Parameters

Parameter	Description
rast	The raster object.
srid	The SRID of the raster object.

Description

If the raster object is stored in external mode, you must make sure that the raster object is geographically referenced and the specified SRID can be found in the spatial_ref_sys table. Otherwise, the information about the raster object cannot be updated.

Examples

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

4.10.18. ST_ScaleX

Queries the pixel width of a raster object on the X scale in the spatial reference system.

Prerequisites

The raster object has a valid spatial reference system identifier (SRID).

Syntax

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

Parameters

Parameter	Description
raster_obj	The name of the raster object.

Examples

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

4.10.19. ST_ScaleY

Queries the pixel width of a raster object on the Y scale in the spatial reference system.

Prerequisites

The raster object has a valid spatial reference system identifier (SRID).

Syntax

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

Parameters

Parameter	Description
raster_obj	The name of the raster object.

Examples

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

4.10.20. ST_SetScale

Configures the pixel widths of a raster object on the X and Y scales in the spatial reference system.

Syntax

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

Parameters

Parameter	Description
raster_obj	The name of the raster object.
scaleX	The pixel width of the raster object on the X scale in the spatial reference system.
scaleY	The pixel width of the raster object on the Y scale in the spatial reference system.
scaleXY	The pixel width of the raster object on the X and Y scales in the spatial reference system. This parameter specifies the same pixel width on the X and Y scales.

Examples

```
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
```

4.10.21. ST_SkewX

Queries the skew of a raster object on the X scale in the spatial reference system.

Prerequisites

The raster object has a valid spatial reference system identifier (SRID).

Syntax

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

Parameters

Parameter	Description
raster_obj	The name of the raster object.

Examples

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

4.10.22. ST_SkewY

Queries the skew of a raster object on the Y scale in the spatial reference system.

Prerequisites

The raster object has a valid spatial reference system identifier (SRID).

Syntax

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

Parameters

Parameter	Description
raster_obj	The name of the raster object.

Examples

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

4.10.23. ST_SetSkew

Configures the skews of a raster object on the X and Y scales in the spatial reference system.

Syntax

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

Parameters

Parameter	Description
raster_obj	The name of the raster object.
skewX	The skew of the raster object on the X scale.

Parameter	Description
skewY	The skew of the raster object on the Y scale.
skewXY	The skew of the raster object on the X and Y scales. This parameter specifies the same skew on the X and Y scales.

Examples

```
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
```

4.10.24. ST_UpperLeftX

Queries the upper-left X coordinate of a raster object in the spatial reference system.

Prerequisites

The raster object has a valid spatial reference system identifier (SRID).

Syntax

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

Parameters

Parameter	Description
raster_obj	The name of the raster object.

Examples

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

4.10.25. ST_UpperLeftY

Queries the upper-left Y coordinate of a raster object in the spatial reference system.

Prerequisites

The raster object has a valid spatial reference system identifier (SRID).

Syntax

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

Parameters

Parameter	Description
raster_obj	The name of the raster object.

Examples

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

4.10.26. ST_SetUpperLeft

Configures the upper-left X and Y coordinates of a raster object in the spatial reference system.

Syntax

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

Parameters

Parameter	Description
raster_obj	The name of the raster object.
upperleftX	The upper-left X coordinate of the raster object in the spatial reference system.
upperleftY	The upper-left Y coordinate of the raster object in the spatial reference system.
upperleftXY	The upper-left X and Y coordinates of the raster object in the spatial reference system. This parameter specifies the same upper-left X and Y coordinates.

Examples

```
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
```

4.10.27. ST_PixelWidth

Queries the pixel width of a raster object in the spatial reference system.

Syntax

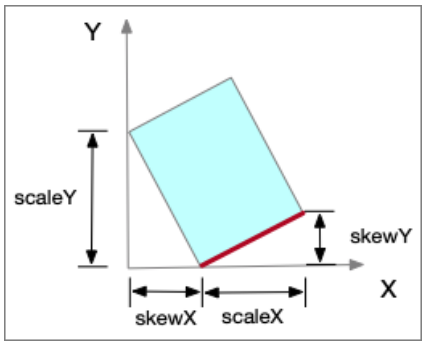
```
float8 ST_PixelWidth(raster raster_obj)
```

Parameters

Parameter	Description
raster_obj	The name of the raster object.

Description

The red part in the following figure shows the pixel width of the raster object. If the skew of the raster object is 0, the pixel width is equal to the X scale.



Examples

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

4.10.28. ST_PixelHeight

Queries the pixel height of a raster object in the spatial reference system.

Syntax

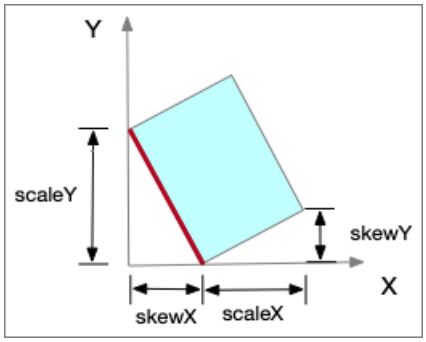
```
float8 ST_PixelHeight(raster raster_obj)
```

Parameters

Parameter	Description
raster_obj	The name of the raster object.

Description

The red part in the following figure shows the pixel height of the raster object. If the skew of the raster object is 0, the pixel height is equal to the Y scale.



Examples

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

4.10.29. ST_Georeference

This function returns the geographic reference information about a raster object. Affine parameters in the text format of "A,B,C,D,E,F" are returned.

Syntax

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

Parameters

Parameter	Description
raster_obj	The raster object.

Examples

```
select ST_Georeference(raster_obj) from raster_table where id=1;
_____________________________________
2.500000000000000,0.000000000000000,38604686.750000000000000,0.000000000000000,-2.500000000000000,4573895.750000000000000
```

4.10.30. ST_IsGeoreferenced

This function returns true if a raster object is geographically referenced. The return value is t or f in Boolean format.

Syntax

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

Parameters

Parameter	Description
raster_obj	The raster object.

Description

In the returned result, t indicates true and f indicates false.

Examples

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

4.10.31. ST_UnGeoreference

This function deletes the geographic reference information about a raster object.

Syntax

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

Parameters

Parameter	Description
rast	The raster object.

Examples

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

4.10.32. ST_SetGeoreference

This function sets the geographic reference information about a raster object.

Syntax

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

Parameters

Parameter	Description
rast	The raster object.
srid	The spatial reference system identifier (SRID) of the raster object.
aop	The spatial reference point. Valid values: • 1: the center cell • 2: the upper-left cell
A, B, C, D, E, and F	The six parameters of an affine transformation. • x = A × Column number + B × Row number + C • y = D × Column number + E × Row number + F

Description

If the raster object is stored in external mode, you need to ensure that the raster object is geographically referenced and the specified SRID can be found in the spatial_ref_sys table. Otherwise, the information about the raster object cannot be updated.

Examples

```
update rast set rast=ST_SetGeoreference(rast,4326,1,8.4163,0,124,0,-8.4163,36.2) where id=1;
_____________________________________
(1 row)
```

4.10.33. ST_RPCGeoreference

Obtains the related information about the rational polynomial coefficients (RPCs) of a raster object. If the specified raster object contains RPC information, a JSON string is returned. If the specified raster object does not contain RPC information, null is returned.

Syntax

```
text ST_RPCGeoreference(raster raster_obj)
```

Parameters

Parameter	Description
raster_obj	The raster object.

Description

The ST_RPCGeoreference function returns a JSON string that contains RPC parameters. This string describes the reference information about the RPC space. The following table describes the supported parameters.

Parameter	Description	Type
lineOff	The line offset.	float
sampOff	The sampling offset.	float
latOff	The latitude offset.	float
longOff	The longitude offset.	float
heightOff	The height offset.	float
lineScale	The line ratio.	float
sampScale	The sampling ratio.	float
latScale	The latitude ratio.	float
longScale	The longitude ratio.	float
heightScale	The height ratio.	float
lineDenCoeff	The number of coefficients for a line denominator. Valid values: 1 to 20.	float array
lineNumCoeff	The number of coefficients for a line numerator. Valid values: 1 to 20.	float array
sampNumCoeff	The number of coefficients for a sample numerator. Valid values: 1 to 20.	float array
sampDenCoeff	The number of coefficients for a sample denominator. Valid values: 1 to 20.	float array
errBias	The bias errors. The root-mean-square (RMS) bias errors in meters. These errors are related to all points per horizontal axis in an image. Default value: `-1.0` .	float

Parameter	Description	Type
errRandom	The random errors. The RMS random errors in meters. These errors are related to each point per horizontal axis in an image. Default value: `-1.0` .	float

Example:

```
SELECT ST_RPCGeoreference(raster_obj) FROM raster_table where id=1;

____________________________________
 {"lineOff":12800.0,"sampOff":4008.0,"latOff":55.0203,"longOff":27.0478,"heightOff":179.0,"lineScale":12800
.0,"sampScale":4008.0,"latScale":0.1238,"longScale":0.0685,"heightScale":300.0,"errBias":-1.0,"errRandom"
:-1.0,"lineDenCoeff":[1.0,-0.00050066513,-0.0014578309,0.0006037474.
.4,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0],"lineNumCoeff":[-0.002104832,-0.01642616,-1.02
7459,0.0041820025,-0.0019027952,0.000016143133,0.00047863558,-0.00021278669,0.0069588307,-0.0000022
605722,-2.2259552e-7,-3.7469372e-7,0.00046486457,-1.8012888e-8,0.0.
.000051407583,0.00075661479,-5.4524409e-7,1.3940799e-7,-0.000018281596,2.4215581e-9],"sampDenCoeff":
[1.0,-0.00050066513,-0.0014578309,0.00060374744,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0],
"sampNumCoeff":[1.0,-0.00050066513,-0.0014578309,0.00060374744,0.0.
.,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0]}
```

4.10.34. ST_SetRPCGeoreference

Specifies rational polynomial coefficient (RPC) parameters for a raster object.

Syntax

```
raster ST_SetRPCGeoreference(raster raster_obj, text rpc_json)
```

Parameters

Parameter	Description
raster_obj	The raster object.
rpc_json	The RPC description in the JSON format.

Description

The following table describes the supported RPC parameters.

Parameter	Description	Type
lineOff	The line offset.	float
sampOff	The sampling offset.	float
latOff	The latitude offset.	float
longOff	The longitude offset.	float

Parameter	Description	Type
heightOff	The height offset.	float
lineScale	The line ratio.	float
sampScale	The sampling ratio.	float
latScale	The latitude ratio.	float
longScale	The longitude ratio.	float
heightScale	The height ratio.	float
lineDenCoeff	The number of coefficients for a line denominator. Valid values: 1 to 20.	float array
lineNumCoeff	The number of coefficients for a line numerator. Valid values: 1 to 20.	float array
sampNumCoeff	The number of coefficients for a sample numerator. Valid values: 1 to 20.	float array
sampDenCoeff	The number of coefficients for a sample denominator. Valid values: 1 to 20.	float array
errBias	The bias errors. The root-mean-square (RMS) bias errors in meters. These errors are related to all points per horizontal axis in an image. Default value: `-1.0`.	float
errRandom	The random errors. The RMS random errors in meters. These errors are related to each point per horizontal axis in an image. Default value: `-1.0`.	float

Example:

```
SELECT ST_RPCGeoreference(ST_setRPCGeoreference(raster_obj, '{
  "lineOff": 12800.00,
  "sampOff": 4008.00,
  "latOff": 55.02030000,
  "longOff": 27.04780000,
  "heightOff": 179.000,
  "lineScale": 12800.00,
  "sampScale": 4008.00,
  "latScale": 0.12380000,
  "longScale": 0.06850000,
  "heightScale": 300.000,
  "lineNumCoeff": [
    -2.104832000000000E-03,
    -1.642616000000000E-02,
    -1.027459000000000E+00,
    4.182002500000000E-03,
    -1.902795200000000E-03,
```

```
    1.614313300000000E-05,
    4.786355800000000E-04,
    -2.127866900000000E-04,
    6.958830700000000E-03,
    -2.260572200000000E-06,
    -2.225955200000000E-07,
    -3.746937200000000E-07,
    4.648645700000000E-04,
    -1.801288800000000E-08,
    5.140758300000000E-06,
    7.566147900000000E-04,
    -5.452440900000000E-07,
    1.394079900000000E-07,
    -1.828159600000000E-05,
    2.421558100000000E-09
  ],
  "lineDenCoeff": [
    1.000000000000000E+00,
    -5.006651300000000E-04,
    -1.457830900000000E-03,
    6.037474400000000E-04,
    0.000000000000000E+00,
    0.000000000000000E+00,
    0.000000000000000E+00,
    0.000000000000000E+00,
    0.000000000000000E+00,
    0.000000000000000E+00,
    0.000000000000000E+00,
    0.000000000000000E+00,
    0.000000000000000E+00,
    0.000000000000000E+00,
    0.000000000000000E+00,
    0.000000000000000E+00,
    0.000000000000000E+00,
    0.000000000000000E+00,
    0.000000000000000E+00
  ],
  "sampNumCoeff": [
    1.000000000000000E+00,
    -5.006651300000000E-04,
    -1.457830900000000E-03,
    6.037474400000000E-04,
    0.000000000000000E+00,
    0.000000000000000E+00,
    0.000000000000000E+00,
    0.000000000000000E+00,
    0.000000000000000E+00,
    0.000000000000000E+00,
    0.000000000000000E+00,
    0.000000000000000E+00,
    0.000000000000000E+00,
    0.000000000000000E+00,
    0.000000000000000E+00,
```

```
      0.000000000000000E+00,
      0.000000000000000E+00,
      0.000000000000000E+00,
      0.000000000000000E+00,
      0.000000000000000E+00
    ],
    "sampDenCoeff": [
      1.000000000000000E+00,
      -5.006651300000000E-04,
      -1.457830900000000E-03,
      6.037474400000000E-04,
      0.000000000000000E+00,
      0.000000000000000E+00,
      0.000000000000000E+00,
      0.000000000000000E+00,
      0.000000000000000E+00,
      0.000000000000000E+00,
      0.000000000000000E+00,
      0.000000000000000E+00,
      0.000000000000000E+00,
      0.000000000000000E+00,
      0.000000000000000E+00,
      0.000000000000000E+00,
      0.000000000000000E+00,
      0.000000000000000E+00,
      0.000000000000000E+00,
      0.000000000000000E+00
    ]
}'))
FROM raster_table
WHERE id = 1;
-------------------------------
 {"lineOff":12800.0,"sampOff":4008.0,"latOff":55.0203,"longOff":27.0478,"heightOff":179.0,"lineScale":12800
.0,"sampScale":4008.0,"latScale":0.1238,"longScale":0.0685,"heightScale":300.0,"errBias":-1.0,"errRandom"
:-1.0,"lineDenCoeff":[1.0,-0.00050066513,-0.0014578309,0.0006037474.
.4,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0]],"lineNumCoeff":[-0.002104832,-0.01642616,-1.02
7459,0.0041820025,-0.0019027952,0.000016143133,0.00047863558,-0.00021278669,0.0069588307,-0.0000022
605722,-2.2259552e-7,-3.7469372e-7,0.00046486457,-1.8012888e-8,0.0.
.000051407583,0.00075661479,-5.4524409e-7,1.3940799e-7,-0.000018281596,2.4215581e-9],"sampDenCoeff":
[1.0,-0.00050066513,-0.0014578309,0.00060374744,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0],
"sampNumCoeff":[1.0,-0.00050066513,-0.0014578309,0.00060374744,0.0.
.,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0]]}
```

4.10.35. ST_NoData

This function returns the predefined NoData value for a band of a raster object. If the NoData value is not defined, the function returns null.

Syntax

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

Parameters

Parameter	Description
raster_obj	The raster object.
band	The band sequence number, starting from 0.

Examples

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

4.10.36. ST_SetNoData

This function sets the NoData value for a specified band of a raster object.

Syntax

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

Parameters

Parameter	Description
rast	The raster object.
band	The sequence number of the band, which starts from 0. A value of -1 indicates all bands.
nodata_value	The NoData value of the band.

Examples

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

4.10.37. ST_ColorTable

This function returns the color table of a band of a raster object in JSON format.

Syntax

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

Parameters

Parameter	Description
raster_obj	The raster object.
band	The sequence number of the band, which starts from 0.

Description

The following code shows color tables in JSON format.

- Four color components:

```
'{"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}
  ]
}'
```

- Three color components:

```
'{"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}
  ]
}'
```

If the band does not have a color table, the function returns null.

Examples

```
select ST_ColorTable(raster_obj,0) from raster_table where id = 1;
_____________________________________
'{"compsCount":3,
  "entries":
  [
    {"value":0,"c1":0,"c2":0,"c3":0},
    {"value":1,"c1":0,"c2":0,"c3":85},
    {"value":2,"c1":0,"c2":0,"c3":170}
  ]
}'
```

4.10.38. ST_SetColorTable

This function sets the color table in JSON format for a specified band of a raster object.

Syntax

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

Parameters

Parameter	Description
rast	The raster object.
band_sn	The sequence number of the band, which starts from 0.
clb	The color table in JSON format. For more information, see the description of the ST_ColorTable function.

Examples

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

4.10.39. ST_Statistics

This function returns the statistics about a band of a raster object in JSON format. If the band does not have statistics, the function returns null.

Syntax

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

Parameters

Parameter	Description
raster_obj	The raster object.
band	The sequence number of the band, which starts from 0.

Examples

```
select ST_Statistics(raster_obj, 0) from raster_table where id=1;
_____________________________________
'{   "min" : 0.00, "max" : 255.00,"mean" : 125.00,"std" : 23.123,"approx" : false}'
```

4.10.40. ST_SetStatistics

This function sets the statistics about a specified band of a raster object.

Syntax

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

Parameters

Parameter	Description
rast	The raster object.
band	The sequence number of the band, which starts from 0.
min,max,mean,std	The statistics.
samplingParams	The sampling parameters, which can be *{"approx":false}* or *{"approx":true}*.

Examples

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

4.10.41. ST_SummaryStats

This topic describes the ST_SummaryStats function. This function is used to calculate the statistics of a specified set of bands that are contained in a raster object.

Syntax

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

Parameters

Parameter	Description
raster_obj	The name of the raster object.
bands	The sequence numbers of the bands. The value of this parameter can be an array of integers or an integer range. The integers start from 0. Examples: `'0'`, `'1-3'`, and `'1,2,3'`.
statsOptions	The JSON string to specify the statistics that you want to calculate.

The following table describes the parameter that is contained in the value of the statsOptions parameter.

Parameter	Description	Type	Format	Default value	Remarks
approx	Specifies whether to use the sampling method to calculate statistics.	boolean	None	true	• true: specifies to use the sampling method to calculate statistics. The statistics may be inaccurate. • false: specifies to calculate all statistics.

Examples:

Calculate the statistics for a specified set of bands that are contained in a raster object.

```
UPDATE raster_obj SET raster_obj=ST_SummaryStats(raster_obj) WHERE id = 1;
UPDATE rast SET rast=ST_SummaryStats(rast,'0-2','{"approx":false}') WHERE id = 1;
UPDATE rast SET rast=ST_SummaryStats(rast,'{"approx":false}') WHERE id = 1;
```

4.10.42. ST_ColorInterp

This function returns the color interpretation type of a band of a raster object.

Syntax

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

Parameters

Parameter	Description
raster_obj	The raster object.
band	The sequence number of the band, which starts from 0.

The following table describes the values of the interp parameter that is returned.

Value	Description
Undefined	The color interpretation type is not defined.
GrayIndex	The index of the associated gray value table.
RGBIndex	The index of the RGB color table.

Value	Description
RGBAIndex	The index of the RGBA color table.
RedBand	The red band in the RGB color model.
GreenBand	The green band in the RGB color model.
BlueBand	The blue band in the RGB color model.
AlphaBand	The alpha band in the RGBA color model.
HueBand	The hue band in the HSL color model.
SaturationBand	The saturation band in the HSL color model.
LightnessBand	The lightness band in the HSL color model.
CyanBand	The cyan band in the CMYK color model.
MagentaBand	The magenta band in the CMYK color model.
YellowBand	The yellow band in the CMYK color model.
BlackBand	The black band in the CMYK color model.
YCbCr_YBand	The Y band in the YCbCr color model.
YCbCr_CbBand	The Cb band in the YCbCr color model.
YCbCr_CrBand	The Cr band in the YCbCr color model.

Example

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

4.10.43. ST_SetColorInterp

This function sets the color interpretation type for a specified band of a raster object.

Syntax

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

Parameters

Parameter	Description
rast	The raster object.

Parameter	Description
band_sn	The sequence number of the band, which starts from 0.
interp	The color interpretation type.

Description

The following table describes the values of the interp parameter.

Value	Description
Undefined	The color interpretation type is not defined.
GrayIndex	The index of the associated gray color table.
RGBIndex	The index of the associated RGB color table.
RGBAIndex	The index of the associated RGBA color table.
CMYKIndex	The index of the associated CMYK color table.
HSLIndex	The index of the associated HSL color table.
RedBand	The red band in the RGB color model.
GreenBand	The green band in the RGB color model.
BlueBand	The blue band in the RGB color model.
AlphaBand	The alpha band in the RGBA color model.
HueBand	The hue band in the HSL color model.
SaturationBand	The saturation band in the HSL color model.
LightnessBand	The lightness band in the HSL color model.
CyanBand	The cyan band in the CMYK color model.
MagentaBand	The magenta band in the CMYK color model.
YellowBand	The yellow band in the CMYK color model.
BlackBand	The black band in the CMYK color model.
YBand	The Y band in the YCbCr color model.
CbBand	The Cb band in the YCbCr color model.
CrBand	The Cr band in the YCbCr color model.

Examples

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

4.10.44. ST_Histogram

This function returns the histogram of a band of a raster object in text format. If the band does not have a histogram, the function returns null.

Syntax

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

Parameters

Parameter	Description
raster_obj	The raster object.
band	The band sequence number, starting from 0.

Examples

```
select ST_Histogram(raster_obj, 0) from raster_table where id=1;
_____________________________________
{
 "approximate":false,
 "histsCounts":
 [2,1,1,0,8,17,47,101,193,345,443,640,877,1189,1560,1847,2087,2560,2816,3193,3567,3840,4101,4415,4498,387
6,3235,2458,1800,1598,1087,731,638,426,264,198,147,126,104,104,80,84,86,71,80,62,74,85,72,80,70,88,69,68,6
2,58,63,51,53,55,54,56,55,63,47,39,49,59,66,62,64,73,66,72,67,84,86,79,91,92,117,138,136,142,157,225,287,285
,382,449,567,628,750,855,1021,1142,1242,1410,1504,1590,1786,1870,2044,2099,2277,2373,2451,2585,2646,28
82,2878,3091,3396,3620,3911,4124,4304,4700,4893,5314,5446,5657,5765,5649,5749,5753,5601,5335,5161,494
3,4592,4445,4207,4083,4090,4270,4465,4514,4844,5204,5331,5597,5777,5838,6004,6316,6095,5762,5567,5465,
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":
   [29.0,30.0,31.0,32.0,33.0,34.0,35.0,36.0,37.0,38.0,39.0,40.0,41.0,42.0,43.0,44.0,45.0,46.0,47.0,48.0,49.0,50.0,5
1.0,52.0,53.0,54.0,55.0,56.0,57.0,58.0,59.0,60.0,61.0,62.0,63.0,64.0,65.0,66.0,67.0,68.0,69.0,70.0,71.0,72.0,73.0,
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,
134.0,135.0,136.0,137.0,138.0,139.0,140.0,141.0,142.0,143.0,144.0,145.0,146.0,147.0,148.0,149.0,150.0,151.0,1
52.0,153.0,154.0,155.0,156.0,157.0,158.0,159.0,160.0,161.0,162.0,163.0,164.0,165.0,166.0,167.0,168.0,169.0,17
0.0,171.0,172.0,173.0,174.0,175.0,176.0,177.0,178.0,179.0,180.0,181.0,182.0,183.0,184.0,185.0,186.0,187.0,188.
0,189.0,190.0,191.0,192.0,193.0,194.0,195.0,196.0,197.0,198.0,199.0,200.0,201.0,202.0,203.0,204.0,205.0,206.0,
207.0]
  }
 }
}
```

4.10.45. ST_SetHistogram

This function sets the histogram in JSON format for a specified band of a raster object.

Syntax

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

Parameters

Parameter	Description
rast	A raster object.
band	The sequence number of the band, which starts from 0.

Parameter	Description
histogram	The histogram in JSON format.

Description

The following table describes the parameters of the histogram in JSON format.

Parameter	Description	Type	Note
approximate	Specifies whether to enable data sampling.	BOOLEAN	None
histsCounts	The array that displays the statistics.	INTEGER[]	None
binFunction/type	The type of the bin function.	STRING	• linear • logarithm • explicit
binFunction/binTable/binValues	The bin values of the bin table.	NUMBER[]	This parameter takes effect only when the type of the bin function is explicit.
binFunction/binRange/minValue	The minimum value of the bin range.	NUMBER	This parameter takes effect only when the type of the bin function is logarithm or linear.
binFunction/binRange/maxValue	The maximum value of the bin range.	NUMBER	This parameter takes effect only when the type of the bin function is logarithm or linear.
binFunction/binRange/outRange	The value beyond the bin range.	NUMBER	This parameter takes effect only when the type of the bin function is logarithm or linear.
binFunction/binRange/binValues	The bin values of the bin range.	NUMBER[]	This parameter takes effect only when the type of the bin function is logarithm or linear.

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

Example

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

4.10.46. ST_BuildHistogram

This function computes the histogram of a band set of a raster object.

Syntax

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

Parameters

Parameter	Description
raster_obj	The raster object.

Examples

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

4.10.47. ST_StatsQuantile

Calculates the quantiles of a raster object.

Syntax

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

Parameters

Parameter	Description
raster_obj	The name of the raster object.

Description

This function calculates quantiles based on bands and records the quantiles in the metadata of the raster object.

Examples

```
UPDATE rat_quantile
SET raster = ST_StatsQuantile(raster)
WHERE id = 1;
```

4.10.48. ST_Quantile

Queries the pixel values of the quantiles for a raster object.

Prerequisites

The quantiles of the raster object are calculated by using the ST_StatsQuantile function.

Syntax

```
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)
```

Parameters

Parameter	Description
raster_obj	The name of the raster object.
quantiles	The quantiles whose pixel values you want to calculate. Valid values: 0.25, 0.5, and 0.75. You can specify one or more quantiles.
bands	The serial numbers of the bands based on which the pixel values of the quantiles are calculated. Supported formats are `'0-2'` and `'1,2,3'`. Serial numbers start from 0. Default value: empty string (`''`). The default value specifies all bands.
exclude_nodata_value	Specifies whether to include NoData values during calculation.
band	Specifies to return the serial numbers of the bands.
quantile	Specifies to return the quantiles.
value	Specifies to return the pixel values.

Examples

```
-- Calculate the pixel value of the 0.25 quantile based on all bands.
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)
-- Calculate the pixel values of the 0.25, 0.5, and 0.75 quantiles based on band 0.
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)
```

4.10.49. ST_MD5Sum

Queries the MD5 hash string of a raster object.

Syntax

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

Parameters

Parameter	Description
raster_obj	The name of the raster object.

Description

The system searches the metadata of the raster object to obtain the MD5 hash string. If the MD5 hash string cannot be found, the system returns a NULL value. Then, the system calculates the MD5 hash string based on the Alibaba Cloud Object Storage Service (OSS) path where the raster object is stored.

The following table describes the parameters that you need to configure.

Parameter	Type	Description
ganos.raster.calculate_md5	boolean	Specifies whether to calculate the MD5 hash string and saves it to the metadata when the raster object is stored. Valid values: true | false. Default value: false. Example: `Set ganos.raster.calculate_md5 = true;`

Parameter	Type	Description
ganos.raster.md5sum_chunk_size	integer	Specifies the size of data that can be cached for each read operation when the system calculates the MD5 hash string. Unit: MB. Valid values: 1 to 256. Default value: 10. Example: `Set ganos.raster.md5sum_chunk_size = 20;`

Examples

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

4.10.50. ST_SetMD5Sum

Configures the MD5 hash string of a raster object.

Syntax

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

Parameters

Parameter	Description
raster_obj	The name of the raster object.
md5sum	The MD5 hash string of the raster object.

Description

This function records the MD5 hash string in the metadata of the raster object. The MD5 hash string must be 32 characters in length and can only contain digits and lowercase letters.

The following table describes the parameters that you need to configure.

Parameter	Type	Description
ganos.raster.calculate_md5	boolean	Specifies whether to calculate the MD5 hash string and saves it to the metadata when the raster object is stored. Valid values: true | false. Default value: false. Example: `Set ganos.raster.calculate_md5 = true;`

Examples

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

4.10.51. ST_XMin

This function returns the minimum X-coordinate value of the specified raster object.

Syntax

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

Parameters

Parameter	Description
raster_obj	The name of the raster object.
pyramid	The pyramid level. The value starts from 0.

Description

If the raster object has been georeferenced, this function returns the minimum X-coordinate value of the raster object. Otherwise, this function returns 0.

Examples

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

4.10.52. ST_YMin

This function returns the minimum Y-coordinate value of the specified raster object.

Syntax

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

Parameters

Parameter	Description
raster_obj	The name of the raster object.
pyramid	The pyramid level. The value starts from 0.

Description

If the raster object has been georeferenced, this function returns the minimum Y-coordinate value of the raster object. Otherwise, this function returns 0.

Examples

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

4.10.53. ST_XMax

This function returns the maximum X-coordinate value of the specified raster object.

Syntax

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

Parameters

Parameter	Description
raster_obj	The name of the raster object.
pyramid	The pyramid level. The value starts from 0.

Description

If the raster object has been georeferenced, this function returns the maximum X-coordinate value of the raster object. Otherwise, this function returns -1.

Examples

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

4.10.54. ST_YMax

This function returns the maximum Y-coordinate value of the specified raster object.

Syntax

```
float8 ST_YMax(raster raster_obj)
```

Parameters

Parameter	Description
raster_obj	The name of the raster object.

Description

If the raster object has been georeferenced, this function returns the maximum Y-coordinate value of the raster object. Otherwise, this function returns -1.

Examples

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

4.10.55. ST_ChunkHeight

This function returns the tile height of the specified raster object.

Syntax

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

Parameters

Parameter	Description
raster_obj	The name of the raster object.

Description

This function returns the tile height of the specified raster object. The tile height is measured in pixels.

Examples

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

4.10.56. ST_ChunkWidth

This function returns the tile width of the specified raster object.

Syntax

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

Parameters

Parameter	Description
raster_obj	The name of the raster object.

Description

This function returns the tile width of the specified raster object. The width is measured in pixels.

Examples

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

4.10.57. ST_ChunkBands

This function returns the number of bands for a tile of the specified raster object.

Syntax

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

Parameters

Parameter	Description
raster_obj	The name of the raster object.

Examples

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

4.10.58. ST_MetaItems

This topic describes the ST_MetaItems function, which returns the customized metadata items of a raster.

Syntax

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

Parameters

Parameter	Description
raster_obj	The raster whose customized metadata items you want to obtain.
band	The sequence number of the band whose customized metadata items you want to obtain. A raster has one or more bands. Valid values start from 0.

Examples

```
-- 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#Conventions,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,_FillValue}
```

4.10.59. ST_SetMetaData

This topic describes the ST_SetMetaData function, which specifies a metadata item of a raster or band.

Syntax

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

Parameters

Parameter	Description
raster_obj	The raster whose metadata item you want to specify.
band	The sequence number of the band whose metadata item you want to specify. Valid values start from 0.
key	The name of the metadata item that you want to specify.
value	The value of the metadata item.

Description

If you specify an empty value (`''`) for a metadata item, the metadata item is deleted.

Examples

```
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
```

4.10.60. ST_BeginDateTime

This topic describes the ST_BeginDateTime function, which obtains the start time of a raster.

Syntax

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

Parameters

Parameter	Description
raster_obj	The raster whose start time you want to obtain.

Examples

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

4.10.61. ST_EndDateTime

This topic describes the ST_EndDateTime function, which obtains the end time of a raster.

Syntax

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

Parameters

Parameter	Description
raster_obj	The raster whose end time you want to obtain.

Examples

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

4.10.62. ST_SetBeginDateTime

This topic describes the ST_SetBeginDateTime function, which specifies the start time of a raster.

Syntax

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

Parameters

Parameter	Description
raster_obj	The raster whose start time you want to specify.

Parameter	Description
time	The start time of the raster. We recommend that you specify the time in the `yyyy-MM-dd HH:mm:ss` format. Example: `2020-08-30 18:00:00`.

Examples

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

4.10.63. ST_SetEndDateTime

This topic describes the ST_SetEndDateTime function, which specifies the end time of a raster.

Syntax

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

Parameters

Parameter	Description
raster_obj	The raster whose end time you want to specify.
time	The end time of the raster. We recommend that you specify the time in the `yyyy-MM-dd HH:mm:ss` format. Example: `2020-08-30 18:00:00`.

Examples

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

4.10.64. ST_DateTime

This topic describes the ST_DateTime function, which obtains the time information of a raster or band.

Syntax

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

Parameters

Parameter	Description
raster_obj	The raster whose time information you want to obtain.
band	The sequence number of the band whose time information you want to obtain. Valid values start from 0.

Description

This function obtains the time information of a raster or band. If you specify the band parameter, this function returns the time information of the specified band. Otherwise, this function returns the time information of all bands of the raster by using a JSON-formatted string.

Examples

```
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 2018","3":"Mon D
ec 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 06:00:0
0 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 D
ec 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":"M
on Dec 31 17:00:00 2018","18":"Mon Dec 31 18:00:00 2018","19":"Mon Dec 31 19:00:00 2018","20":"Mon Dec 3
1 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"
```

4.10.65. ST_SetDateTime

This topic describes the ST_SetDateTime function, which specifies the start and end time of a raster or the time information of a band.

Syntax

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

Parameters

Parameter	Description
raster_obj	The raster whose start and end time you want to specify.
band	The sequence number of the band whose start and end time you want to specify. Valid values start from 0.
time	The time information of the specified band. We recommend that you specify the time in the `yyyy-MM-dd HH:mm:ss` format. Example: `2020-08-30 18:00:00`.
start	The start time that you want to specify. We recommend that you specify the time in the `yyyy-MM-dd HH:mm:ss` format.
end	The end time that you want to specify. We recommend that you specify the time in the `yyyy-MM-dd HH:mm:ss` format.

Examples

```
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
```

4.11. Algebra and Analysis Functions

4.11.1. ST_Reclassify

The ST_Reclassify function returns a raster object. The returned object has the same spatial reference and resolution as the original image. The number of bands is specified by the relassexpr parameter.

Syntax

```
raster ST_Reclassify(raster raster_obj,
        cstring reclassexpr default NULL
        cstring storageOption default '')
```

Parameters

Parameter	Description
raster_obj	The raster object to be reclassified.
reclassexpr	The JSON string is used to indicate a numerical interval for classification.
storageOption	The storage option that is represented by a JSON string in the returned result.

The reclassexpr parameter indicates a JSON string array. Each child JSON object indicates a band operation parameter. The following table describes the parameters.

Parameter	Description	Type	Default value	Notes
band	The band sequence number.	integer	None	The band sequence number starts from 0.
remap	The parameter that is used for classification.	object	-	-
nodata	Specifies whether to use nodata.	boolean	false	• If you set this parameter to true, the pixel value is nodata and the classification result is also nodata. • If you set this parameter to false, non-nodata values are calculated as general numeric values.
nodataValue	The nodata value.	float8	0	The new nodata value.

The remap parameter specifies how to map original pixel values to new pixel values.

- The key indicates the original pixel value range and consists of one or more numeric values that are separated with commas (,). You can specify an open or closed range at the start and end.
  - A left parenthesis "(" indicates greater than.
  - A right parenthesis ")" indicates less than.
  - A right bracket "]" indicates less than or equal to.
  - A left bracket "[" indicates greater than or equal to.

  The default value is `()`.
- The value indicates the result of the mapping from original to new pixel values and consists of one or more numeric values that are separated with commas (,).
- Three mapping methods are available:
  - Range-to-range mapping: The original and new pixel ranges have the same number of numeric values. Examples: "300,400,500":"80,90,100" and "[300,400,500]":"80,90,100".
  - Range-to-value mapping: The original pixel range has one more numeric value than the new pixel range. Example: "(300,400,500]":"80,90".
  - Value-to-value mapping: The original and new pixel ranges both have only one numeric value. Example: "10":"1".
- If a pixel value does not belong to a mapping range, the pixel value is considered as a nodata value.
- Multiple ranges cannot contain the same pixel value.
- Examples

- Example 1

  The following example shows how to reclassify band 0:

```
if 0<old_value<=100
  new_value = 20
else if 100<old_value<=200
  new_value = 50
else
  new_value = 0
```

```
[
 {
   "band":0,
   "remap":{
      "(0,100,200]":"20,50"
   }
 }
]
```

- Example 2

  Multiple segmented values are supported. Pixel values beyond the range are set to nodata values.

```
[
 {
   "band":0,
   "remap":{
      "(0,100,200]":"20,50",
      "(300,400,500]":"80,90,100"
   }
 }
]
```

- Example 3

  The following example shows how to reclassify band 0:

```
if 0<old_value<=100
  new_value = 20
else if 100<old_value<=200
  new_value = 50
else
  new_value = 999
```

  The following example shows how to reclassify band 1:

```
if 400<old_value<=600
  new_value = 20+(old_value-400)/200 * (90-20)
else if 600<old_value<=800
  new_value = 90+(old_value-600)/200 * (130-90)
else
  new_value = 0
```

```
[
 {
  "band":0,
  "remap":{
   "(0,100,200]":"20,50"
  },
  "nodata":true,
  "nodataValue":999
 },
 {
  "band":1,
  "remap":{
   "(400,600,800]":"20,90,130"
  },
  "nodata":false,
  "nodataValue":0
 }
]
```

The following table describes the parameters of storageOption.

Parameter	Description	Type	Default value	Notes
chunking	Specifies whether to store data as chunks.	boolean	Same as the original raster	-
chunkdim	The dimension information about the chunk.	string	Same as the original raster	This parameter only takes effect when the chunking parameter is set to true.

Parameter	Description	Type	Default value	Notes
chunktable	The name of the chunk table.	string	''	If the '' value is passed, a temporary chunk table that has a random name is generated to store data. This temporary table is valid in only the current session. If you need to retain an accessible clipping object, you must specify the name of the chunk table.
compression	The type of the compression algorithm.	string	Same as the original raster	Only none, jpeg, zlib, png, lzo, and lz4 are supported.
quality	The compression quality.	integer	Same as the original raster	This parameter only takes effect for the jpeg compression algorithm.
interleaving	The interleaving method.	string	Same as the original raster	The value must be one of the following types: • bip: band interleaved by pixel (BIP) • bil: band interleaved by line (BIL) • bsq: band sequential (BSQ)
endian	The endian.	string	Same as the original raster	The value must be one of the following types: • NDR: little endian • XDR: big endian
celltype	The pixel type.	string	Same as the original raster	-

Examples

```
-- Create a permanent table.
CREATE TABLE rast_reclassify_result(id integer, rast raster);
-- Create a temporary table.
CREATE TEMP TABLE rast_reclassify_result_temp(id integer, rast raster);
-- Store the result in a temporary table.
INSERT INTO rast_reclassify_result_temp(id, rast)
select 1, ST_Reclassify(rast, '[{"band":0,"remap":{"(0,100,200]":"20,50"}}]')
from reclass_table
```

4.11.2. ST_MapAlgebra

This topic describes the ST_MapAlgebra function. This function uses algebraic expressions in compliance with Algebra Computing Language to transform multiple original raster objects into one raster object.

Syntax

```
raster ST_MapAlgebra(raster[] rasters ,
    cstring algebraExpr  default NULL,
    cstring storageoption default '')
```

Parameters

Parameter	Description
rasters	The original raster objects that you want to transform.
algebraExpr	The JSON string that specifies the algebraic expressions that you want to use.
storageOption	The JSON string that specifies how to store the new raster object.

> **Note** The system only requires that the lengths and widths of the original raster objects are in the same unit of measurement. It does not check their spatial reference systems or resolutions. If the lengths and widths of the original raster objects are in different units of measurement, you can use the ST_Tranform, ST_Resize, and ST_Clip functions to convert them into the same unit.

Each child JSON object in the JSON string specified by the algebraExpr parameter represents a field dictated to an algebraic expression. The following table describes these fields.

Field	Description	Type	Default value	Setting notes
algebraExpr	The algebraic expression that you want to use.	String	N/A	N/A.
nodata	Specifies whether nodata values are valid.	Boolean	false	• If you set this field to true, nodata values are valid and pixels with nodata values are not transformed. • If you set this field to false, nodata values are invalid and pixels with nodata values are transformed.
nodataValue	The nodata value to return for the new raster object.	float8	0	N/A.

The algebraic expression specified by the algebraExpr field contains the following keywords:

- [r, b]
  - r: indicates the ID of the new raster object in the array specified by the rasters parameter. Format: 0-n-1.
  - b: indicates the band number of the new raster object. Format: 0-n-1.
- x

  The sequence number of the column where the specified pixel resides.
- y

  The sequence number of the row where the specified pixel resides.

The following table lists the operations that you can incorporate in an algebraic expression.

Type	Operator or function	Remarks
Operators	• + • - • * • / • % (remainder) • ** (power)	N/A.
Bitwise operations	• << • >> • & • | • ^	N/A.
Logical operations	• < • > • == • ! = • <= • >= • && • || • !	N/A.

Type	Operator or function	Remarks
Operations on functions	• abs • sqrt • exp • log • ln • sin • cos • tan • sinh • cosh • tanh • arcsin • arccos • arctan • ceil • floor • round	You can specify only one operation on a function.
Statistical functions	• min • max • sum • mean • majority • minority • std • median • range • variety	You must specify two or more statistical functions.

- Example 1

  Transform the original raster object into a new raster object that only has one band. The return result is as follows: raster[0]band[0] + raster[1]band[0] * raster[1]band[1].

```
[
  {
    "expr":"([0,0] + [1,0] * [1,1]) ",
    "nodata": true,
    "nodataValue":999
  }
]
```

- Example 2

  Compute the variance of three bands.

```
[
  {
    "expr":"(std([0,0],[0,1],[0,2]))",
    "nodata": true,
    "nodataValue":999
  }
]
```

- Example 3

  Transform the original raster objects into a new raster object that has three bands. Each band is transformed by using a unique expression.

```
[
  {
    "expr":"(min([0,0],[0,1],[0,2]))",
    "nodata": true,
    "nodataValue":999
  },
  {
    "expr":"(max([0,0],[0,1],[0,2]))",
    "nodata": true,
    "nodataValue":999
  },
  {
    "expr":"(mean([0,0],[0,1],[0,2]))",
    "nodata": true,
    "nodataValue":999
  }
]
```

The following table describes fields in the storageOption parameter.

Field	Description	Type	Default value	Setting notes
chunking	Specifies whether to store the new raster object as chunks.	Boolean	Same as the original raster object	N/A.
chunkdim	The dimensions used to store the new raster object as chunks.	String	Same as the original raster object	This field only takes effect when the chunking field is set to true.

Field	Description	Type	Default value	Setting notes
chunktable	The name of the chunk table.	String	Null string ('')	By default, a temporary chunk table with a random name is generated to store data. This temporary chunk table is only valid in the current session. To save the new raster object permanently, you must specify you want to create a permanent chunk table in the chunktable field.
compression	The format used for image compression.	String	Same as the original raster object	Six compression formats are supported: None, JPEG, Zlib, PNG, LZO, and LZ4.
quality	The image quality of the new raster object.	Integer	Same as the original raster object	This field only takes effect in JPEG format.
interleaving	The interleaving type of the new raster object.	String	Same as the original raster object	Valid values: • bip: band interleaved by pixel (BIP) • bil: band interleaved by line (BIL) • bsq: band sequential (BSQ)
endian	The endian format of the new raster object.	String	Same as the original raster object	Valid values: • NDR: specifies little endian format. • XDR: specifies big endian format.

Examples

```
-- Create a permanent chunk table.
CREATE TABLE rast_mapalgebra_result(id integer, rast raster);
-- Insert data into a chunk table.
WITH foo AS (
 SELECT 1 AS rid, rast AS rast from t1 WHERE id = 1
UNION ALL
 SELECT 2 AS rid, rast AS rast from t2 WHERE id = 2
)
INSERT INTO rast_mapalgebra_result
SELECT 1, ST_MapAlgebra(
  ARRAY(SELECT rast FROM foo ORDER BY rid),
  '[{"expr":"([0,0] + 0.5 * [1,0] - ([1,1])","nodata": true, "nodataValue":999}]',
  '{"chunktable":"algebra_rbt"}'
);
```

4.12. Raster Image Processing

4.12.1. ST_SubRaster

This topic describes the ST_SubRaster function. This function transforms a pyramid level or band of a raster object.

Syntax

```
raster ST_SubRaster(raster raster_obj ,
      integer pyramidLevel default 0,
      cstring bands default '', /* All bands */
      cstring storageOption  default '')
```

Parameters

Parameter	Description
raster_obj	The original raster object whose pyramid level or band you want to transform.
pyramidLevel	The pyramid level that you want to transform.
bands	The band that you want to transform. The value of this field can be an array of integers or an integer range starting from 0. Examples: "0-2" and "1,2,3'. Default value: null string (''). The default value specifies to transform all bands.
storageOption	The JSON string that specifies how to store the new raster object.

The following table describes fields in the storageOption parameter.

Field	Description	Type	Default value	Setting notes

Field	Description	Type	Default value	Setting notes
chunking	Specifies whether to store the new raster object as chunks.	Boolean	Same as the original raster object	N/A.
chunkdim	The dimensions used to store the new raster object as chunks.	String	Same as the original raster object	This field only takes effect when the chunking field is set to true.
chunktable	The name of the chunk table.	String	Null string ('')	By default, a temporary chunk table with a random name is generated to store data. This temporary chunk table is only valid in the current session. To save the new raster object permanently, you must specify you want to create a permanent chunk table in the chunktable field.
compression	The format used for image compression.	String	Same as the original raster object	Six compression formats are supported: None, JPEG, Zlib, PNG, LZO, and LZ4.
quality	The image quality of the new raster object.	Integer	Same as the original raster object	This field only takes effect in JPEG format.
interleaving	The interleaving type of the new raster object.	String	Same as the original raster object	Valid values: • bip: band interleaved by pixel (BIP) • bil: band interleaved by line (BIL) • bsq: band sequential (BSQ)
endian	The endian format of the new raster object.	String	Same as the original raster object	Valid values: • NDR: specifies little endian format. • XDR: specifies big endian format.

Examples

```
SELECT ST_SubRaster(rast, 1, '0-2', '{"chunktable":"chunk_table", "chunking":true}')
FROM raster_sub
WHERE id=1;
```

4.12.2. ST_Transform

This topic describes the ST_Transform function. This function transforms the underlying coordinates of a raster object from one known spatial reference system to another.

Syntax

```
raster ST_Transform(raster rast,
      integer outSrid,
      cstring processexpr default '',
      cstring storageOption default '')
```

Parameters

Parameter	Description
rast	The original raster object whose coordinates you want to transform.
outSrid	The spatial reference system of the new raster object. The value of this parameter must be a valid SIRD that can be queried from the spatial_ref_sys table.
processExpr	The JSON string that specifies how to resample pixels and process nodata values.
storageOption	The JSON string that specifies how to store the new raster object.

Each child JSON object in the JSON string specified by the processExpr parameter represents a field. The following table describes these fields.

Field	Description	Type	Default value	Setting notes
resample	The method used to resample pixels.	Text	Near	Valid values: Near | Average | Cubic | Bilinear.
nodata	Specifies whether nodata values from the original raster object are valid.	Boolean	false	• If you set this field to true, nodata values are valid and pixels with nodata values are not resampled. • If you set this field to false, nodata values are invalid and pixels with nodata values are resampled.

Field	Description	Type	Default value	Setting notes
nodataValue	The new nodata value specified based on the bands in the new raster object.	float8 float8[]	NULL	You can specify one or more nodata values in the nodataValue field. • If you specify one nodata value, this nodata value is used for all bands in the new raster object. • If you specify more than one nodata value, the number of specified nodata values must be the same as the number of bands in the new raster object.

Note Exercise caution when specifying the nodata and nodataValue fields. If the original raster object does not have pixels with nodata values, we recommend that you set the nodata field to false and do not specify the nodataValue field. Otherwise, image artifacts may occur.

The following table describes fields in the storageOption parameter.

Field	Description	Type	Default value	Setting notes
chunking	Specifies whether to store the new raster object as chunks.	Boolean	Same as the original raster object	N/A.
chunkdim	The dimensions used to store the new raster object as chunks.	String	Same as the original raster object	This field only takes effect when the chunking field is set to true.
chunktable	The name of the chunk table.	String	Null string ('')	By default, a temporary chunk table with a random name is generated to store data. This temporary chunk table is only valid in the current session. To save the new raster object permanently, you must specify you want to create a permanent chunk table in the chunktable field.

Field	Description	Type	Default value	Setting notes
compression	The format used for image compression.	String	Same as the original raster object	Six compression formats are supported: None, JPEG, Zlib, PNG, LZO, and LZ4.
quality	The image quality of the new raster object.	Integer	Same as the original raster object	This field only takes effect in JPEG format.
interleaving	The interleaving type of the new raster object.	String	Same as the original raster object	Valid values: • bip: band interleaved by pixel (BIP) • bil: band interleaved by line (BIL) • bsq: band sequential (BSQ)
endian	The endian format of the new raster object.	String	Same as the original raster object	Valid values: • NDR: specifies little endian format. • XDR: specifies big endian format.

Note If you set the chunktable field to NULL or a null string (''), a temporary chunk table with a random name is generated in the current session to store the new raster object. The temporary chunk table is only valid in the current session, and is deleted immediately after the current session ends. To save the new raster object permanently, you must specify you want to create a permanent chunk table in the chunktable field.

Examples

```
CREATE TABLE if not exists datasource_table(id integer, rast raster);
INSERT INTO datasource_table values(1, ST_ImportFrom('rbt','$(RAST_DATA_DIR)/512_512_1_bsq_8u_geo.tif', '{}'));
-----------------------------------------------------
-- Method 1: Specify the chunktable field to create a permanent chunk table for storing the new raster object.
-----------------------------------------------------
CREATE TABLE rat_transform_result(id integer, rast raster);
-- If you do not specify the nodata field:
INSERT INTO rat_transform_result(id, rast)
select 10, ST_Transform(rast,32652, '{"resample":"Near","nodata":false}','{"chunking":true,"chunkdim":"(256,256,1)","compression":"none","interleaving":"bsq","chunktable":"result_rbt"}')
from datasource_table
where id =1;
-- If you specify one nodata value in the nodataValue field and pixels with nodata values are resampled:
INSERT INTO rat_transform_result(id, rast)
select 11, ST_Transform(rast,32652, '{"resample":"Near","nodata":true,"nodatavalue":255}','{"chunking":true,"chunkdim":"(256,256,1)","compression":"none","interleaving":"bsq","chunktable":"result_rbt"}')
from datasource_table
where id =1;
-- If you specify more than one nodata value in the nodataValue field:
INSERT INTO rat_transform_result(id, rast)
select 12, ST_Transform(rast,32652, '{"resample":"Near","nodata":false,"nodatavalue":[255,255,255]}','{"chunking":true,"chunkdim":"(256,256,1)","compression":"none","interleaving":"bsq","chunktable":"result_rbt"}')
from datasource_table
where id =1;
-----------------------------------------------------
-- Method 2: Do not specify the chunktable field. The new raster object is saved to a temporary chunk table with a random name and can only be used for nested computations in the current session.
-----------------------------------------------------
CREATE TEMP TABLE rat_transform_result_temp(id integer, rast raster);
INSERT INTO rast_clip_result_temp(id, rast)
select 1, ST_Transform(rast,32652,'{"resample":"Near","nodata":false, "nodataValue":[255,255,255]}','{"chunking":true,"chunkdim":"(256,256,1)","compression":"none","interleaving":"bsq"}')
from datasource_table
where id =1;
```

4.12.3. ST_Rescale

This topic describes the ST_Rescale function. This function rescales the pixel range of a raster object but leaves the geospatial data range unchanged.

Syntax

```
raster ST_Rescale(raster rast,
        f8 scale_x,
        f8 scale_y,
         cstring processexpr default '',
         cstring storageOption default '');
raster ST_Rescale(raster raster,
        f8 scale_xy,
         cstring processexpr default '',
         cstring storageOption default '')
```

Parameters

Parameter	Description
rast	The original raster object whose pixel range you want to rescale.
scale_x	The proportional ratio to rescale the original raster object in the x direction.
scale_y	The proportional ratio to rescale the original raster object in the y direction.
scale_xy	The proportional ratio to rescale the original raster object in both the x and y directions.
processExpr	The JSON string that specifies how to resample pixels and process nodata values.
storageOption	The JSON string that specifies how to store the new raster object.

Each child JSON object in the JSON string specified by the processExpr parameter represents a field. The following table describes these fields.

Field	Description	Type	Default value	Setting notes
resample	The method used to resample pixels.	text	Near	Valid values: Near | Average | Cubic | Bilinear.
nodata	Specifies whether nodata values from the original raster object are valid.	Boolean	false	• If you set this field to true, nodata values are valid and pixels with nodata values are not resampled. • If you set this field to false, nodata values are invalid and pixels with nodata values are resampled.

Field	Description	Type	Default value	Setting notes
nodataValue	The new nodata value specified based on the bands in the new raster object.	float8 float8[]	NULL	You can specify one or more nodata values in the nodataValue field. • If you specify one nodata value, this nodata value is used for all bands in the new raster object. • If you specify more than one nodata value, the number of specified nodata values must be the same as the number of bands in the new raster object.

Note Exercise caution when you specify the nodata and nodataValue fields. If the original raster object does not have pixels with nodata values, we recommend that you set the nodata field to false and do not specify the nodataValue field. Otherwise, image artifacts may occur.

The following table describes fields in the storageOption parameter.

Field	Description	Type	Default value	Setting notes
chunking	Specifies whether to store the new raster object as chunks.	Boolean	Same as the original raster object	N/A.
chunkdim	The dimensions used to store the new raster object as chunks.	String	Same as the original raster object	This field only takes effect when the chunking field is set to true.
chunktable	The name of the chunk table.	String	Null string ('')	By default, a temporary chunk table with a random name is generated to store data. This temporary chunk table is only valid in the current session. To save the new raster object permanently, you must specify you want to create a permanent chunk table in the chunktable field.

Field	Description	Type	Default value	Setting notes
compression	The format used for image compression.	String	Same as the original raster object	Six compression formats are supported: None, JPEG, Zlib, PNG, LZO, and LZ4.
quality	The image quality of the new raster object.	Integer	Same as the original raster object	This field only takes effect in JPEG format.
interleaving	The interleaving type of the new raster object.	String	Same as the original raster object	Valid values: • bip: band interleaved by pixel (BIP) • bil: band interleaved by line (BIL) • bsq: band sequential (BSQ)
endian	The endian format of the new raster object.	String	Same as the original raster object	Valid values: • NDR: specifies little endian format. • XDR: specifies big endian format.

Note If you set the chunktable field to NULL or a null string (''), a temporary chunk table with a random name is generated in the current session to store the new raster object. The temporary chunk table is only valid in the current session, and is deleted immediately after the current session ends. To save the new raster object permanently, you must specify you want to create a permanent chunk table in the chunktable field.

Examples

```
CREATE TABLE if not exists datasource_table(id integer, rast raster);
INSERT INTO datasource_table values(1, ST_ImportFrom('rbt','$(RAST_DATA_DIR)/512_512_1_bsq_8u_geo.tif', '{}'));
-----------------------------------------------------
-- Method 1: Specify the chunktable field to create a permanent chunk table to store the new raster object.
-----------------------------------------------------
CREATE TABLE rat_rescale_result(id integer, rast raster);
-- If you do not specify the nodata field:
INSERT INTO rat_rescale_result(id, rast)
select 10, ST_Rescale(rast,1024,1024, '{"resample":"Near","nodata":false}','{"chunking":true,"chunkdim":"(256,256,1)","compression":"none","interleaving":"bsq","chunktable":"result_rbt"}')
from datasource_table
where id =1;
-- If you specify one nodata value in the nodataValue field and pixels with nodata values are resampled:
INSERT INTO rat_rescale_result(id, rast)
select 11, ST_Rescale(rast,1024,1024, '{"resample":"Near","nodata":true,"nodatavalue":255}','{"chunking":true,"chunkdim":"(256,256,1)","compression":"none","interleaving":"bsq","chunktable":"result_rbt"}')
from datasource_table
where id =1;
-- If you specify more than one nodata value in the nodataValue field:
INSERT INTO rat_rescale_result(id, rast)
select 12, ST_Rescale(rast,1024,1024, '{"resample":"Near","nodata":false,"nodatavalue":[255,255,255]}','{"chunking":true,"chunkdim":"(256,256,1)","compression":"none","interleaving":"bsq","chunktable":"result_rbt"}')
from datasource_table
where id =1;
-----------------------------------------------------
-- Method 2: Do not specify the chunktable field. The new raster object is saved to a temporary chunk table with a random name and can only be used for nested computations in the current session.
-----------------------------------------------------
CREATE TEMP TABLE rat_rescale_result_temp(id integer, rast raster);
INSERT INTO rat_rescale_result_temp(id, rast)
select 1, ST_Rescale(rast,1024,1024,'{"resample":"Near","nodata":false, "nodataValue":[255,255,255]}','{"chunking":true,"chunkdim":"(256,256,1)","compression":"none","interleaving":"bsq"}')
from datasource_table
where id =1;
```

4.12.4. ST_Resize

This topic describes the ST_Resize function. This function resizes the pixel range of a raster object but leaves the geospatial data range unchanged.

Syntax

```
raster ST_Resize(raster rast,
        integer outWidth,
        integer outHeight,
         cstring processexpr default '',
         cstring storageOption default '')
```

Parameters

Parameter	Description
rast	The original raster object whose pixel range you want to resize.
outWidth	The width of the new raster object in pixels.
outHeight	The height of the new raster object in pixels.
processExpr	The JSON string that specifies how to resample pixels and process nodata values.
storageOption	The JSON string that specifies how to store the new raster object.

Each child JSON object in the JSON string specified by the processExpr parameter represents a field. The following table describes these fields.

Field	Description	Type	Default value	Setting notes
resample	The method used to resample pixels.	text	Near	Valid values: Near | Average | Cubic | Bilinear.
nodata	Specifies whether nodata values from the original raster object are valid.	Boolean	false	• If you set this field to true, nodata values are valid and pixels with nodata values are not resampled. • If you set this field to false, nodata values are invalid and pixels with nodata values are resampled.
nodataValue	The new nodata value specified based on the bands in the new raster object.	float8 float8[]	NULL	You can specify one or more nodata values in the nodataValue field. • If you specify one nodata value, this nodata value is used for all bands in the new raster object. • If you specify more than one nodata value, the number of specified nodata values must be the same as the number of bands in the new raster object.

Note Exercise caution when you specify the nodata and nodataValue fields. If the original raster object does not have pixels with nodata values, we recommend that you set the nodata field to false and do not specify the nodataValue field. Otherwise, image artifacts may occur.

The following table describes fields in the storageOption parameter.

Field	Description	Type	Default value	Setting notes
chunking	Specifies whether to store the new raster object as chunks.	Boolean	Same as the original raster object	N/A.
chunkdim	The dimensions used to store the new raster object as chunks.	String	Same as the original raster object	This field only takes effect when the chunking field is set to true.
chunktable	The name of the chunk table.	String	Null string ('')	By default, a temporary chunk table with a random name is generated to store data. This temporary chunk table is only valid in the current session. To save the new raster object permanently, you must specify you want to create a permanent chunk table in the chunktable field.
compression	The format used for image compression.	String	Same as the original raster object	Six compression formats are supported: None, JPEG, Zlib, PNG, LZO, and LZ4.
quality	The image quality of the new raster object.	Integer	Same as the original raster object	This field only takes effect in JPEG format.
interleaving	The interleaving type of the new raster object.	String	Same as the original raster object	Valid values: • bip: band interleaved by pixel (BIP) • bil: band interleaved by line (BIL) • bsq: band sequential (BSQ)
endian	The endian format of the new raster object.	String	Same as the original raster object	Valid values: • NDR: specifies little endian format. • XDR: specifies big endian format.

> **Note** If you set the chunktable field to NULL or a null string (''), a temporary chunk table with a random name is generated in the current session to store the new raster object. The temporary chunk table is only valid in the current session, and is deleted immediately after the current session ends. To save the new raster object permanently, you must specify you want to create a permanent chunk table in the chunktable field.

Examples

```
CREATE TABLE if not exists datasource_table(id integer, rast raster);
INSERT INTO datasource_table values(1, ST_ImportFrom('rbt','$(RAST_DATA_DIR)/512_512_1_bsq_8u_geo.tif', '{}'));
-----------------------------------------------------
-- Method 1: Specify the chunktable field to create a permanent chunk table to store the new raster object.
-----------------------------------------------------
CREATE TABLE rat_resize_result(id integer, rast raster);
-- If you do not specify the nodata field:
INSERT INTO rat_resize_result(id, rast)
select 10, ST_Resize(rast,1024,1024, '{"resample":"Near","nodata":false}','{"chunking":true,"chunkdim":"(256,256,1)","compression":"none","interleaving":"bsq","chunktable":"result_rbt"}')
from datasource_table
where id =1;
-- If you specify one nodata value in the nodataValue field and pixels with nodata values are resampled:
INSERT INTO rat_resize_result(id, rast)
select 11, ST_Resize(rast,1024,1024, '{"resample":"Near","nodata":true,"nodatavalue":255}','{"chunking":true,"chunkdim":"(256,256,1)","compression":"none","interleaving":"bsq","chunktable":"result_rbt"}')
from datasource_table
where id =1;
-- If you specify more than one nodata value in the nodataValue field:
INSERT INTO rat_resize_result(id, rast)
select 12, ST_Resize(rast,1024,1024, '{"resample":"Near","nodata":false,"nodatavalue":[255,255,255]}','{"chunking":true,"chunkdim":"(256,256,1)","compression":"none","interleaving":"bsq","chunktable":"result_rbt"}')
from datasource_table
where id =1;
-----------------------------------------------------
-- Method 2: Do not specify the chunktable field. The new raster object is saved to a temporary chunk table with a random name and can only be used for nested computations in the current session.
-----------------------------------------------------
CREATE TEMP TABLE rat_resize_result_temp(id integer, rast raster);
INSERT INTO rat_resize_result_temp(id, rast)
select 1, ST_Resize(rast,1024,1024,'{"resample":"Near","nodata":false, "nodataValue":[255,255,255]}','{"chunking":true,"chunkdim":"(256,256,1)","compression":"none","interleaving":"bsq"}')
from datasource_table
where id =1;
```

4.12.5. ST_RPCRectify

Rectifies a raster data file based on the related rational polynomial coefficients (RPC) parameters and returns the updated raster data file.

Syntax

```
raster ST_RPCRectify(raster rast,
 raster dem default NULL,
            cstring rectifyOption default '{}',
            cstring storageOption default '{}',
  cstring parallelOption default '{}')
```

Parameters

Parameter	Description
rast	The raster object to be rectified.
dem	A digital elevation model (DEM) that is referenced when this function rectifies the raster object. Default value: null.
rectifyOption	The options to be rectified. The value is a JSON-formatted string.
storageOption	The storage options for the output. The value is a JSON-formatted string.
parallelOption	The parallel options. The value is a JSON-formatted string.

Description

> **Note** The raster object must contain RPC information.

rectifyOption is an array of strings in the JSON format. The following table describes the parameters of each JSON object.

Parameter	Description	Type	Default value	Description
resample	The method that is used to resample pixels.	string	'Near'	Valid values: • Near • Average • Cubic • Bilinear
outSrid	The spatial reference information of the new raster object.	integer	The value is the same as the value for the original raster object.	None

Parameter	Description	Type	Default value	Description
nodataValue	The new nodata value that is specified based on the bands of the new raster object.	float8 float8[]	NULL	You can specify a value or an array in the nodataValue parameter. • If you specify a value, the value is used for all bands in the new raster object. • If you specify an array [0, 0, 0], the number of array elements must be the same as the number bands in the new raster object. • If you do not specify this parameter, the system sets this parameter to the value of the nodataValue parameter for the original raster object. If the original nodataValue parameter is not specified, the system sets this parameter to 0.
RPC_DEM_MISSING_VALUE	The height value when the dem parameter is set to the new nodata value or the raster data is not included in the valid values of the dem parameter.	float8	0	None

The following table describes the parameters of the storageOption parameter.

Parameter	Description	Type	Default value	Description
chunking	Specifies whether to store the data as chunks.	boolean	The value is the same as the value for the original raster object.	None

Parameter	Description	Type	Default value	Description
chunkdim	The dimensions that are used to chunk data.	string	The value is the same as the value for the original raster object.	This parameter takes effect only when you set the chunking parameter to true.
chunktable	The name of the chunk table.	string	''	If you specify the value '', the system create a temporary chunk table that has a random name in the current session. This chunk table is used to store the new raster object. The temporary chunk table is available only in the current session. After the session ends, the temporary chunk table is also deleted. If you want to save the new raster object, you must specify a name for the chunk table. If the specified parallel method does not support chunk tables that are created by anonymous users, you must create a chunk table in advance.
compression	The format into which you want to compress the data of the raster object.	string	The value is the same as the value for the original raster object.	Valid values: • NONE • JPEG • ZLIB • PNG • LZO • LZ4 • JP2K
quality	The quality of compression.	integer	The value is the same as the value for the original raster object.	This parameter is available only for JPEG and compression algorithms.

Parameter	Description	Type	Default value	Description
interleaving	The method that is used to interleave the data of the raster object.	string	The value is the same as the value for the original raster object.	Valid values: • bip: band interleaved by pixel (BIP) • bil: band interleaved by line (BIL) • bsq: band sequential (BSQ)
endian	The endian format of the new raster object.	string	The value is the same as the value for the original raster object.	Valid values: • NDR: little endian • XDR: big endian

Examples:

```
CREATE TABLE if not exists raster_table(id integer, rast raster);
-- The RPC raster data.
INSERT INTO raster_table values(1, ST_ImportFrom('t_chunk','<RAST_DATA_DIR>/rpc.tif', '{}'));
-- The DEM data.
INSERT INTO raster_table values(2, ST_ImportFrom('t_chunk','<RAST_DATA_DIR>/rpc_dem.tif', '{}'));
CREATE TABLE raster_result(id integer, rast raster);
-- The dem parameter is not specified. The system samples data by using the Nearest Neighbor Search (NNS) method.
INSERT INTO raster_result(id, rast)
select 10, ST_RPCRectify(rast,
                NULL,
                 '{"resample":"Near"}',
                 '{"chunking":true,"chunkdim":"(256,256,1)","interleaving":"bsq","chunktable":"t_chunk"}')
from raster_table
where id =1;
-- The dem parameter is specified. The system samples data by using the NNS method.
INSERT INTO raster_result(id, rast)
select 11, ST_RPCRectify(rast,
                (SELECT rast FROM raster_table WHERE id = 2),
                 '{"resample":"Near", "outSrid": 32635}',
                 '{"chunking":true,"chunkdim":"(256,256,1)","interleaving":"bsq","chunktable":"t_chunk"}',
                 '{"parallel": 8}')
from raster_table
where id =1;
```

4.13. Operators

4.13.1. Equal to operator (=)

This topic describes the equal to operator (=). This operator determines whether the universally unique identifier (UUID) of the first specified raster object is equal to the UUID of the second.

Syntax

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

Parameters

Parameter	Description
rast1	The raster object 1.
rast2	The raster object 2.

Note This operator only compares the UUIDs of two specified raster objects. This operator does not compare their spatial extents or pixel types. This operator is used only for operations such as union and B-tree.

Examples

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

4.13.2. Greater than operator (>)

This topic describes the greater than operator (>). This operator determines whether the universally unique identifier (UUID) of the first specified raster object is greater than the UUID of the second.

Syntax

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

Parameters

Parameter	Description
rast1	The raster object 1.
rast2	The raster object 2.

> **Note** This operator only compares the UUIDs of two specified raster objects. This operator does not compare their spatial extents or pixel types. This operator is used only for operations such as union and B-tree.

Examples

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

4.13.3. Less than operator (<)

This topic describes the less than operator (<). This operator determines whether the universally unique identifier (UUID) of the first specified raster object is less than the UUID of the second.

Syntax

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

Parameters

Parameter	Description
rast1	The raster object 1.
rast2	The raster object 2.

> **Note** This operator only compares the UUIDs of two specified raster objects. This operator does not compare their spatial extents or pixel types. This operator is used only for operations such as union and B-tree.

Examples

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

4.13.4. Greater than or equal to operator (>=)

This topic describes the greater than or equal to operator (>=). This operator determines whether the universally unique identifier (UUID) of the first specified raster object is greater than or equal to the UUID of the second.

Syntax

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

Parameters

Parameter	Description
rast1	The raster object 1.
rast2	The raster object 2.

> **Note** This operator only compares the UUIDs of two specified raster objects. It does not compare their spatial extents or pixel types. This operator is used only for operations such as union and B-tree.

Example

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

4.13.5. Less than or equal to operator (<=)

This topic describes the less than or equal to operator (<=). This operator determines whether the universally unique identifier (UUID) of the first specified raster object is less than or equal to the UUID of the second.

Syntax

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

Parameters

Parameter	Description
rast1	The raster object 1.
rast2	The raster object 2.

> **Note** This operator only compares the UUIDs of two specified raster objects. This operator does not compare their spatial extents or pixel types. This operator is used only for operations such as union and B-tree.

Examples

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

4.14. Functions for Identifying Spatial Relationships

4.14.1. ST_Intersects

This topic describes the ST_Intersects function. This function identifies the spatial relationship between two raster objects or between a raster object and a geometric object to determine whether the first specified object intersects with the second.

Syntax

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

Parameters

Parameter	Description
rast1	The raster object 1.
rast2	The raster object 2.
rast	A raster object.
geom	A geometric object.

Example

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

4.14.2. ST_Contains

This topic describes the ST_Contains function. This function identifies the spatial relationship between two raster objects or between a raster object and a geometric object to determine whether the first specified object completely contains the second.

Syntax

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

Parameters

Parameter	Description
rast1	The raster object 1.
rast2	The raster object 2.
rast	A raster object.
geom	A geometric object.

Example

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

4.14.3. ST_ContainsProperly

This topic describes the ST_ContainsProperly function. This function identifies the spatial relationship between two raster objects or between a raster object and a geometric object to determine whether the first specified object properly contains the second.

Syntax

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

Parameters

Parameter	Description
rast1	The raster object 1.
rast2	The raster object 2.
rast	A raster object.
geom	A geometric object.

Example

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

4.14.4. ST_Covers

This topic describes the ST_Covers function. This function identifies the spatial relationship between two raster objects or between a raster object and a geometric object to determine whether the first specified object covers the second.

Syntax

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

Parameters

Parameter	Description
rast1	The raster object 1.
rast2	The raster object 2.
rast	A raster object.
geom	A geometric object.

Example

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

4.14.5. ST_CoveredBy

This topic describes the ST_CoveredBy function. This function identifies the spatial relationship between two raster objects or between a raster object and a geometric object to determine whether the first specified object is covered by the second.

Syntax

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

Parameters

Parameter	Description
rast1	The raster object 1.
rast2	The raster object 2.
rast	A raster object.

Parameter	Description
geom	A geometric object.

Example

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

4.14.6. ST_Disjoint

This topic describes the ST_Disjoint function. This function identifies the spatial relationship between two raster objects or between a raster object and a geometric object to determine whether the first specified object does not touch or intersect the second.

Syntax

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

Parameters

Parameter	Description
rast1	The raster object 1.
rast2	The raster object 2.
rast	A raster object.
geom	A geometric object.

Example

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

4.14.7. ST_overlaps

This topic describes the ST_overlaps function. This function identifies the spatial relationship between two raster objects or between a raster object and a geometric object to determine whether the first specified object overlaps with the second.

Syntax

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

Parameters

Parameter	Description
rast1	The raster object 1.
rast2	The raster object 2.
rast	A raster object.
geom	A geometric object.

Example

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

4.14.8. ST_Touches

This topic describes the ST_Touches function. This function identifies the spatial relationship between two raster objects or between a raster object and a geometric object to determine whether the first specified object touches the second.

Syntax

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

Parameters

Parameter	Description
rast1	The raster object 1.
rast2	The raster object 2.
rast	A raster object.
geom	A geometric object.

Example

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

4.14.9. ST_Within

This topic describes the ST_Within function. This function identifies the spatial relationship between two raster objects or between a raster object and a geometric object to determine whether the first specified object is completely within the second.

Syntax

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

Parameters

Parameter	Description
rast1	The raster object 1.
rast2	The raster object 2.
rast	A raster object.
geom	A geometric object.

Example

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

4.15. Auxiliary functions

4.15.1. ST_CreateChunkTable

This topic describes the ST_CreateChunkTable function. This function is used to create a database chunk table.

Syntax

```
boolean ST_CreateChunkTable(cstring tableName)
```

Parameters

Parameter	Description
tableName	The name of the database chunk table. The name must meet the following requirements: • The name must start with a letter and end with a letter or digit. • The name can contain lowercase letters, digits, and underscores (_). • The name must be 2 to 64 characters in length. • The name must be unique in the current database.

Examples:

```
SELECT ST_CreateChunkTable('t_chunk');
_____________________________________
t
```

4.15.2. ST_CheckGPU

This function checks whether a GPU is available in the current environment.

Syntax

```
text ST_CheckGPU();
```

Description

This function checks whether a GPU can be identified in the current runtime environment of ApsaraDB for RDS.

Examples

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

4.15.3. ST_AKId

Queries the AccessKey ID that is used to access a raster object stored in Alibaba Cloud Object Storage Service (OSS). This function can be used with the function that is used to modify multiple AccessKey IDs at a time.

Prerequisites

The raster object is stored in OSS.

Syntax

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

Parameters

Parameter	Description
raster_obj	The name of the raster object.

Examples

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

4.15.4. ST_SetAccessKey

Configures the logon information that is used to access a raster object stored in Alibaba Cloud Object Storage Service (OSS).

Prerequisites

The raster object is stored in OSS.

Syntax

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

Parameters

Parameter	Description
raster_obj	The name of the raster object.
id	The AccessKey ID that is used to log on to OSS. If you set this parameter to NULL, the previous AccessKey ID is used.
key	The AccessKey secret that is used to log on to OSS. If you set this parameter to NULL, the previous AccessKey secret is used.
valid	Specifies whether to verify the validity of the logon information.

Examples

```
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
```

4.15.5. ST_SetAKId

Sets the AccessKey ID that is used to access a raster object stored in Alibaba Cloud Object Storage Service (OSS).

Prerequisites

The raster object is stored in OSS.

Syntax

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

Parameters

Parameter	Description
raster_obj	The name of the raster object.
id	The AccessKey ID that is used to log on to OSS. If you set this parameter to NULL, the previous AccessKey ID is used.
valid	Specifies whether to verify the validity of the logon information.

Examples

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

4.15.6. ST_SetAKSecret

Sets the AccessKey secret that is used to access a raster object stored in Alibaba Cloud Object Storage Service (OSS).

Prerequisites

The raster object is stored in OSS.

Syntax

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

Parameters

Parameter	Description
raster_obj	The name of the raster object.
key	The AccessKey secret that is used to log on to OSS. If you set this parameter to NULL, the previous AccessKey secret is used.
valid	Specifies whether to verify the validity of the logon information.

Examples

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

4.15.7. ST_RasterDrivers

This topic describes the ST_RasterDrivers function. This function is used to query all drivers that are supported by raster data sources in Ganos.

Syntax

```
setof record ST_RasterDrivers(out idx integer,
                out short_name text,
               out long_name text,
               out can_read boolean,
                out can_export boolean,
               out can_asfile boolean,
                out create_options text);
```

Parameters

Parameter	Description
idx	The ID of the driver.
short_name	The abbreviated name of the driver.
long_name	The full name of the driver.
can_read	Valid values: • true: The file formats that are supported by the driver can be used in the ST_CreateRast and ST_ImportFrom functions. For more information, see ST_CreateRast and ST_ImportFrom. • false: The file formats that are supported by the driver cannot be used in the ST_CreateRast or ST_ImportFrom function. If you use these file formats in the ST_CreateRast or ST_ImportFrom function, errors are reported. For more information, see ST_CreateRast and ST_ImportFrom.

Parameter	Description
can_export	Valid values: • true: The file formats that are supported by the driver can be used in the ST_ExportTo function. For more information, see ST_ExportTo. • false: The file formats that are supported by the driver cannot be used in the ST_ExportTo function. If you use these file formats in the ST_ExportTo function, errors are reported. For more information, see ST_ExportTo.
can_asfile	Valid values: • true: The file formats that are supported by the driver can be used in the ST_AsDatasetFile function. For more information, see ST_AsDatasetFile. • false: The file formats that are supported by the driver cannot be used in the ST_AsDatasetFile function. If you use these file formats in the ST_AsDatasetFile function, errors are reported. For more information, see ST_AsDatasetFile.
create_options	The options that are included in the create_option parameter in the ST_ExportTo and ST_AsDatasetFile functions. For more information, see ST_ExportTo and ST_AsDatasetFile.

Examples

```
-- Query information about the netCDF driver.
SELECT * FROM
st_rasterdrivers()
where short_name='netCDF';
 idx | short_name |      long_name       | can_read | can_export | can_asfile |
create_options
-----+------------+--------------------------+----------+------------+-----------
 36 | netCDF     | Network Common Data Format | t      | t          | t
| <CreationOptionList>  <Option name='FORMAT' type='string-select'
default='NC'>    <Value>NC</Value>    <Value>NC2</Value>
   <Value>NC4</Value>    <Value>NC4C</Value>  </Option>  <Option
name='COMPRESS' type='string-select' default='NONE'>    <Value>NONE</Value>
<Value>DEFLATE</Value>  </Option>  <Option name='ZLEVEL' ty
pe='int' description='DEFLATE compression level 1-9' default='1'/>  <Option
name='WRITE_BOTTOMUP' type='boolean' default='YES'>  </Option>  <Option
name='WRITE_GDAL_TAGS' type='boolean' default='YES'>  </Opt
ion>  <Option name='WRITE_LONLAT' type='string-select'>    <Value>YES</Value>
<Value>NO</Value>    <Value>IF_NEEDED</Value>  </Option>  <Option
name='TYPE_LONLAT' type='string-select'>    <Value>float<
/Value>    <Value>double</Value>  </Option>  <Option name='PIXELTYPE'
type='string-select' description='only used in Create()'>
<Value>DEFAULT</Value>      <Value>SIGNEDBYTE</Value>  </Option>  <Opti
on name='CHUNKING' type='boolean' default='YES' description='define chunking
when creating netcdf4 file'/>  <Option name='MULTIPLE_LAYERS' type='string-
select' description='Behaviour regarding multiple vector l
ayer creation' default='NO'>      <Value>NO</Value>
<Value>SEPARATE_FILES</Value>      <Value>SEPARATE_GROUPS</Value>  </Option>
<Option name='CONFIG_FILE' type='string' description='Path to a XML con
figuration file (or content inlined)'/></CreationOptionList>
-- Query all drivers that are supported by the ST_ImportFrom and ST_CreateRast functions.
select short_name from
st_rasterdrivers()
where can_read =true;
-- Query all drivers that are supported by the ST_ExportTo function.
select short_name from
st_rasterdrivers()
where can_export =true;
-- Query all drivers that are supported by the ST_AsDatasetFile function.
select short_name from
st_rasterdrivers()
where can_asfile =true;
```

4.16. Variables

4.16.1. ganos.parallel.transaction

This topic describes the ganos.parallel.transaction variable. This variable specifies whether Ganos can commit and roll parallel transactions back along with the main transaction while parallel operations are run.

Data type

String

Value

- **transaction_commit**: specifies that Ganos can commit or roll parallel transactions back along with the main transaction. This is the default value.
- **fast_commit**: specifies that Ganos cannot commit or roll parallel transactions back along with the main transaction.

Examples:

```
SET ganos.parallel.transaction = transaction_commit;
```

4.16.2. ganos.parallel.degree

This topic describes the ganos.parallel.degree variable. If you do not specify the degree of parallelism, Ganos uses the value of this variable as the default degree of parallelism. You can set the default parallelism by using the ganos.parallel.degree variable.

Data type

Integer

Value

Valid values: 1 to 64. Default value: 1.

Examples:

```
SET ganos.parallel.degree = 4;
```

4.16.3. ganos.raster.calculate_md5

This topic describes the ganos.raster.calculate_md5 variable. This variable specifies whether PolarDB calculates the MD5 hash string and saves the string to the metadata when a raster object is imported.

Data type

Boolean

Value

- **true**: specifies that PolarDB calculates the MD5 hash string and saves the string to the metadata when a raster object is imported.
- **false**: specifies that PolarDB does not calculate the MD5 hash string when a raster object is imported. This is the default value.

Example

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

4.16.4. ganos.raster.md5sum_chunk_size

This topic describes the ganos.raster.md5sum_chunk_size variable. This variable specifies the amount of data that can be cached for each read operation when PolarDB calculates the MD5 hash string.

Data type

Integer

Value

Valid values: 10 to 100. Unit: MB. Default value: 10.

Example

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

4.16.5. ganos.raster.mosaic_must_same_nodata

This topic describes the ganos.raster.mosaic_must_same_nodata variable. This variable specifies whether the NoData values in the data source must be the same during the Mosaic operation.

Data type

Boolean

Value

- **true**: specifies that the NoData values in the data source must be the same. This is the default value.
- **false**: specifies that the NoData values in the data source can be different.

> **Note** PolarDB does not convert the NoData values during the mosaic operation. If you set this parameter to **false**, the semantics of the pixels may change after the mosaic operation.

Example

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

5.SpatialRef SQL reference

5.1. ST_SrEqual

This topic describes the ST_SrEqual function, which determines whether two spatial reference systems are the same.

Syntax

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

Parameters

Parameter	Description
sr1	The string representing spatial reference system 1. It must be an OGC WKT or PROJ.4 string.
sr2	The string representing spatial reference system 2. It must be an OGC WKT or PROJ.4 string.
strict	Specifies whether to use the strict comparison method. The value true specifies to compare the names of the reference ellipsoids if the spatial reference systems are georeferenced. Default value: true.

Description

This function parses the semantics of two spatial reference systems to compare the parameter information of these systems. The compared parameter information includes the projection methods, the reference ellipsoids, and the long and short axes of the reference ellipsoids. If the spatial reference systems are the same, this function returns t. Otherwise, this function returns f.

Example:

```
-- Compare two text-based spatial reference systems.
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
 -- Search for the spatial reference system identifier (SRID) of a spatial reference system from the spatial_ref_sys table.
 select srid from spatial_ref_sys where st_srequal(srtext::cstring, '+proj=longlat +ellps=GRS80 +no_defs ') limit 1;
 srid
------
 3824
```

5.2. ST_SrReg

This topic describes the ST_SrReg function, which registers a spatial reference system.

Syntax

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

Parameters

Parameter	Description
sr	The string representing the spatial reference system. It must be an OGC WKT or PROJ.4 string.
auth_name	The author who defines the spatial reference system. Example: EPSG.
auth_id	The spatial reference system identifier (SRID) of the spatial reference system.

Description

If the spatial reference system already exists, this function returns the SRID of the existing spatial reference system. If the spatial reference system does not exist, this function inserts a record into the spatial_ref_sys table and returns the SRID of the new spatial reference system.

Example:

```
-- Register a spatial reference system that already exists.
select 4490, ST_srReg('GEOGCS["China Geodetic Coordinate System 2000",DATUM["China_2000",SPHEROID
["CGCS2000",6378137,298.257222101,AUTHORITY["EPSG","1024"]],AUTHORITY["EPSG","1043"]],PRIMEM["G
reenwich",0,AUTHORITY["EPSG","8901"]],UNIT["degree",0.0174532925199433,AUTHORITY["EPSG","9122"]],
AUTHORITY["EPSG","4490"]]');
 st_srreg
----------
    4490
-- Register a new spatial reference system.
select ST_srReg('user_defined',100, 'GEOGCS["User Geodetic Coordinate System ",DATUM["China_2000",SP
HEROID["CGCS2000",6378137,298.257222101,AUTHORITY["EPSG","903"]],AUTHORITY["EPSG","1043"]],PRIM
EM["Greenwich",0,AUTHORITY["EPSG","8901"]],UNIT["degree",0.0174532925199433,AUTHORITY["EPSG","9
122"]],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 +towgs84=24.4
7,-130.89,-81.56,0,0,0.13,-0.22 +units=m +no_defs');
 st_srreg
----------
  10002
```

5.3. ST_SrFromEsriWkt

This function converts the specified string from the Esri well-known text (WKT) format to the Open Geospatial Consortium (OGC) WKT format.

Syntax

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

Parameters

Parameter	Description
sr1	The spatial reference string that uses the Esri WKT format.

Examples

```
SELECT ST_srFromEsriWkt('GEOGCS["China Geodetic Coordinate System 2000",DATUM["D_China_2000",SPHEROID["CGCS2000",6378137,298.257222101]],PRIMEM["Greenwich",0],UNIT["Degree",0.0174532925199433295]]');
st_srfromesriwkt
----------
 GEOGCS["China Geodetic Coordinate System 2000",DATUM["China_2000",SPHEROID["CGCS2000",6378137,298.257222101]],PRIMEM["Greenwich",0],UNIT["Degree",0.017453292519943295]]
```

6.Point cloud SQL reference

6.1. Constructors

6.1.1. ST_makePoint

This function constructs a pcpoint object.

Syntax

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

Parameters

Parameter	Description
pcid	The ID of the schema, which comes from the point_cloud_formats table.
float8[]	The float8 array. The number of entries in the array is equal to the number of dimensions specified in the schema.

Examples

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

6.1.2. ST_makePatch

This function constructs a pcpatch object.

Syntax

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

Parameters

Parameter	Description
pcid	The ID of the schema, which comes from the pointcloud_formats table.
float8[]	The float8 array. The number of entries in the array is an integral multiple of the number of dimensions specified in the schema.

Examples

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

6.1.3. ST_Patch

This function constructs a pcpatch object from a pcpoint array.

Syntax

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

Parameters

Parameter	Description
pts	The pcpoint array.

Examples

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

6.2. Attribute functions

6.2.1. ST_asText

This function converts a pcpoint or pcpatch object into a JSON-formatted string.

Syntax

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

Parameters

Parameter	Description
pp	The pcpatch object.
pt	The pcpoint object.

Examples

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

6.2.2. ST_pcID

This function returns the schema ID of a pcpoint or pcpatch object.

Syntax

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

Parameters

Parameter	Description
pt	The pcpoint object.
pp	The pcpatch object.

Examples

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

6.2.3. ST_get

This function returns the values of attribute dimensions in a pcpoint object.

Syntax

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

Parameters

Parameter	Description
pc	The pcpoint object.
dimname	The name of the specified attribute dimension.

Examples

```
SELECT ST_Get('010100000064CEFFFF941100007030000000400'::pcpoint, 'Intensity');
------------------
4
SELECT ST_Get('010100000064CEFFFF941100007030000000400'::pcpoint);
--------------------
{-127,45,124,4}
```

6.2.4. ST_numPoints

This function returns the number of pcpoint objects in a pcpatch object.

Syntax

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

Parameters

Parameter	Description
pc	The pcpatch object.

Examples

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

6.2.5. ST_summary

This function returns the summary of a pcpatch object.

Syntax

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

Parameters

Parameter	Description
pc	The pcpatch object.

Examples

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

6.3. Object operations

6.3.1. ST_boundingDiagonalGeometry

This function returns the diagonal of the bounding box of a pcpatch object as a geometry object.

Syntax

```
geometry ST_boundingDiagonalGeometry(pcpatch pc);
```

Parameters

Parameter	Description
pc	The pcpatch object.

Description

The diagonal of a bounding box is a 2D linestring object of Ganos Geometry. This function can be used to create an index on a pcpatch object column.

Examples

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

6.3.2. ST_compress

This function compresses a pcpatch object based on the specified compression method.

Syntax

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

Parameters

Parameter	Description
pc	The pcpatch object.
global_compression_schema	The compression schema.
compression_config	The compression configuration item that specifies the compression algorithm for specific dimensions.

Description

Valid values of the global_compression_schema parameter include:

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

If the global_compression_schema parameter is set to dimension, valid values of the compression_config parameter include:

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

Examples

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

6.3.3. ST_explode

This function separates a pcpatch object into pcpoint objects in multiple rows.

Syntax

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

Parameters

Parameter	Description
pc	The pcpatch object.

Examples

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

6.3.4. ST_patchAvg

This function computes the average values of all the attribute dimensions for all the pcpoint objects in a pcpatch object and returns a new pcpoint object whose attribute dimensions are set to these average values.

Syntax

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

Parameters

Parameter	Description
pc	The pcpatch object.

Examples

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

6.3.5. ST_envelopeGeometry

This function returns the bounding box of a pcpatch object as a geometry object.

Syntax

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

Parameters

Parameter	Description
pc	The pcpatch object.

Description

A bounding box is a 2D geometry object of Ganos Geometry.

Examples

```
SELECT ST_AsText(ST_EnvelopeGeometry(pa)) FROM patches LIMIT 1;
------------------------------
POLYGON((-126.99 45.01,-126.99 45.09,-126.91 45.09,-126.91 45.01,-126.99 45.01))
CREATE INDEX ON patches USING GIST(ST_EnvelopeGeometry(patch));
```

6.3.6. ST_filterGreaterThan

This function filters all pcpoint objects to obtain the pcpoint objects whose values of the specified attribute dimension are greater than a fixed value in a pcpatch object and returns a new pcpatch object that is composed of these pcpoint objects.

Syntax

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

Parameters

Parameter	Description
pc	The pcpatch object.
dimname	The name of the specified attribute dimension.
value	The fixed value of the attribute dimension.

Examples

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

6.3.7. ST_filterLessThan

This function filters all pcpoint objects to obtain the pcpoint objects whose values of the specified attribute dimension are smaller than a fixed value in a pcpatch object and returns a new pcpatch object that is composed of these pcpoint objects.

Syntax

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

Parameters

Parameter	Description
pc	The pcpatch object.
dimname	The name of the specified attribute dimension.
value	The fixed value of the attribute dimension.

Examples

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

6.3.8. ST_filterEquals

This function filters all pcpoint objects to obtain the pcpoint objects whose values of the specified attribute dimension are equal to a fixed value in a pcpatch object and returns a new pcpatch object that is composed of these pcpoint objects.

Syntax

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

Parameters

Parameter	Description
pc	The pcpatch object.
dimname	The name of the specified attribute dimension.

Parameter	Description
value	The fixed value of the attribute dimension.

Examples

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

6.3.9. ST_filterBetween

This function filters all pcpoint objects to obtain the pcpoint objects whose values of the specified attribute dimension are between two fixed values in a pcpatch object and returns a new pcpatch object that is composed of these pcpoint objects.

Syntax

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

Parameters

Parameter	Description
pc	The pcpatch object.
dimname	The name of the specified attribute dimension.
minvalue	The minimum fixed value of the attribute dimension.
maxvalue	The maximum fixed value of the attribute dimension.

Description

The pcpoint objects whose values of the specified attribute dimension are equal to the minimum or maximum fixed value are not returned.

Examples

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

6.3.10. ST_isSorted

This function returns true if all the pcpoint objects in a pcpatch object are sorted based on the specified attribute dimensions.

Syntax

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

Parameters

Parameter	Description
pc	The pcpatch object.
dimnames	The array of attribute dimension names.
strict	Indicates whether each pcpoint object has unique values for the specified attribute dimensions.

Examples

```
SELECT ST_isSorted(pa, Array['a','c']) FROM patches;
-------------------------------------------------------------
f
(1 rows)
```

6.3.11. ST_patchMax

This function computes the maximum values of all the attribute dimensions for all the pcpoint objects in a pcpatch object and returns a new pcpoint object whose attribute dimensions are set to these maximum values.

Syntax

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

Parameters

Parameter	Description
pc	The pcpatch object.

Examples

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

6.3.12. ST_patchMin

This function computes the minimum values of all the attribute dimensions for all the pcpoint objects in a pcpatch object and returns a new pcpoint object whose attribute dimensions are set to these minimum values.

Syntax

```
numeric ST_patchMin(pcpatch pc);
```

Parameters

Parameter	Description
pc	The pcpatch object.

Examples

```
SELECT ST_AsText(ST_PatchMin(pa)) FROM patches WHERE id = 7;
---------------------------------
{"pcid":1,"pt":[-126.5,45.5,50,5]}
```

6.3.13. ST_patchAvg

This function computes the average value of an attribute dimension for all the pcpoint objects in a pcpatch object.

Syntax

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

Parameters

Parameter	Description
pc	The pcpatch object.
dimname	The name of the specified attribute dimension.

Examples

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

6.3.14. ST_patchMax

This function computes the maximum value of an attribute dimension for all the pcpoint objects in a pcpatch object.

Syntax

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

Parameters

Parameter	Description
pc	The pcpatch object.
dimname	The name of the specified attribute dimension.

Examples

```
SELECT ST_PatchMax(pa, 'intensity') FROM patches WHERE id = 7;
--------------------------
125.0000000000000000
```

6.3.15. ST_patchMin

This function computes the minimum value of an attribute dimension for all the pcpoint objects in a pcpatch object.

Syntax

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

Parameters

Parameter	Description
pc	The pcpatch object.
dimname	The name of the specified attribute dimension.

Examples

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

6.3.16. ST_pointN

This function returns the pcpoint object with the specified sequence number in a pcpatch object.

Syntax

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

Parameters

Parameter	Description
pc	The pcpatch object.
n	The sequence number of the pcpoint object, starting from 1. If n is a negative number, it specifies the pcpoint object with the sequence number of |n| starting from the end of the pcpatch object reversely. For example, a value of -2 indicates the last but one pcpoint object in the pcpatch object.

Examples

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

6.3.17. ST_sort

This function sorts all the pcpoint objects in a pcpatch object based on the specified attribute dimensions and returns a new pcpatch object that is composed of the sorted pcpoint objects.

Syntax

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

Parameters

Parameter	Description
pc	The pcpatch object.
dimnames	The array of attribute dimension names.

Examples

```
update patches set pa=ST_Sort(pa, Array['y','x']);
---------------------------------------
(1 rows)
```

6.3.18. ST_range

This function obtains n consecutive pcpoint objects that start from a sequence number from a pcpatch object and returns a new pcpatch object that is composed of these pcpoint objects.

Syntax

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

Parameters

Parameter	Description
pc	The pcpatch object.
start	The sequence number of the start pcpoint object. The value starts from 1.
n	The number of pcpoint objects after the pcpoint object (included) specified by the start parameter.

Examples

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

6.3.19. ST_setPcid

This function sets a new schema for a pcpatch object and returns a new pcpatch object.

Syntax

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

Parameters

Parameter	Description
pc	The pcpatch object.
pcid	The ID of the new schema, which comes from the pointcloud_formats table.
def	The specified value. Attribute dimensions that exist in the new schema but not in the old schema are set to this value. The default value is 0.0.

Description

Attribute dimensions that exist in the old schema but not in the new schema are discarded.

Examples

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

6.3.20. ST_transform

This function transforms a pcpatch object based on a new schema and returns a new pcpatch object.

Syntax

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

Parameters

Parameter	Description
pc	The pcpatch object.
pcid	The ID of the new schema, which comes from the pointcloud_formats table.
def	The specified value. Attribute dimensions that exist in the new schema but not in the old schema are set to this value. The default value is 0.0.

Description

Different from the ST_setPcid function, the ST_transform function returns a new pcpatch object based on the different dimension interpretations, scales, or offsets in the new schema.

Examples

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

6.3.21. ST_unCompress

This function returns an uncompressed version of the pcpatch object.

Syntax

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

Parameters

Parameter	Description
pc	The pcpatch object.

Description

All the pcpatch objects returned by functions are compressed based on the compression method specified in the schema before they are returned.

Examples

```
SELECT ST_Uncompress(pa) FROM patches WHERE ST_NumPoints(pa) = 1;
---------------------------------------
0101000000000000000001000000C8CEFFFFF8110000102700000A00
```

6.3.22. ST_union

This function merges an array of pcpatch entries into a pcpatch object.

Syntax

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

Parameters

Parameter	Description
pcs	The pcpatch array.

Examples

```
-- Compare npoints(sum(patches)) with sum(npoints(patches)).
SELECT ST_NumPoints(ST_Union(pa)) FROM patches;
SELECT Sum(ST_NumPoints(pa)) FROM patches;
100
```

6.4. OGC WKB operations

6.4.1. ST_asBinary

This function returns the well-known binary (WKB) value of a pcpoint object.

Syntax

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

Parameters

Parameter	Description
pt	The pcpoint object.

Examples

```
SELECT ST_AsBinary('010100000064CEFFFF941100000703000000400'::pcpoint);
--------------------------------------------
\x01010000800000000000c05fc0000000000080464000000000000005f40
```

6.4.2. ST_envelopeAsBinary

This function returns the well-known binary (WKB) value of the bounding box of a pcpatch object.

Syntax

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

Parameters

Parameter	Description
pc	The pcpatch object.

Description

A bounding box is a 2D geometry object of Ganos Geometry.

Examples

```
SELECT ST_EnvelopeAsBinary(pa) FROM patches LIMIT 1;
--------------------------------------------
\x0103000000010000000500000090c2f5285cbf5fc0e17a
14ae4781464090c2f5285cbf5fc0ec51b81e858b46400ad7
a3703dba5fc0ec51b81e858b46400ad7a3703dba5fc0e17a
14ae4781464090c2f5285cbf5fc0e17a14ae47814640
```

6.4.3. ST_boundingDiagonalAsBinary

This function returns the well-known binary (WKB) value of the bounding box diagonal of a pcpatch object.

Syntax

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

Parameters

Parameter	Description
pc	The pcpatch object.

Description

A bounding box is a 2D geometry object.

Examples

```
SELECT ST_BoundingDiagonalAsBinary( ST_Patch(ARRAY[ ST_MakePoint(1, ARRAY[0.,0.,0.,10.]), ST_MakePoint(1, ARRAY[1.,1.,1.,10.]), ST_MakePoint(1, ARRAY[10.,10.,10.,10.])]));
------------------------------------------------------------
\x01020000a0e610000002000000000000000000000000000000000000000000000000000000000000000000000000000000024400000000000024400000000000002440
```

6.5. Spatial relationship judgment

6.5.1. ST_intersects

This function returns true if the bounding boxes of two pcpatch objects or the bounding boxes of a pcpatch object and a geometry object intersect.

Syntax

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

Parameters

Parameter	Description
pp1	Pcpatch object 1.
pp2	Pcpatch object 2.
g	The geometry object of Ganos Geometry.

Examples

```
-- Patch should intersect itself
SELECT ST_Intersects(
    '0101000000000000000001000000C8CEFFFFF8110000102700000A00'::pcpatch,
    '0101000000000000000001000000C8CEFFFFF8110000102700000A00'::pcpatch);
------------------------
t
SELECT ST_Intersects('SRID=4326;POINT(-126.451 45.552)'::geometry, pa) FROM patches WHERE id = 7;
------------------------
t
```

6.6. Spatial processing

6.6.1. ST_intersection

This function returns a new pcpatch object that indicates the intersection of a pcpatch object and the given geometry object.

Syntax

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

Parameters

Parameter	Description
pc	The pcpatch object.
geom	The geometry object of Ganos Geometry.

Examples

```
SELECT ST_AsText(ST_Explode(ST_Intersection(
    pa,
    'SRID=4326;POLYGON((-126.451 45.552, -126.42 47.55, -126.40 45.552, -126.451 45.552))'::geometry
)))
FROM patches WHERE id = 7;
         st_astext
---------------------------------------
{"pcid":1,"pt":[-126.44,45.56,56,5]}
{"pcid":1,"pt":[-126.43,45.57,57,5]}
{"pcid":1,"pt":[-126.42,45.58,58,5]}
{"pcid":1,"pt":[-126.41,45.59,59,5]}
```

7.Trajectory SQL reference

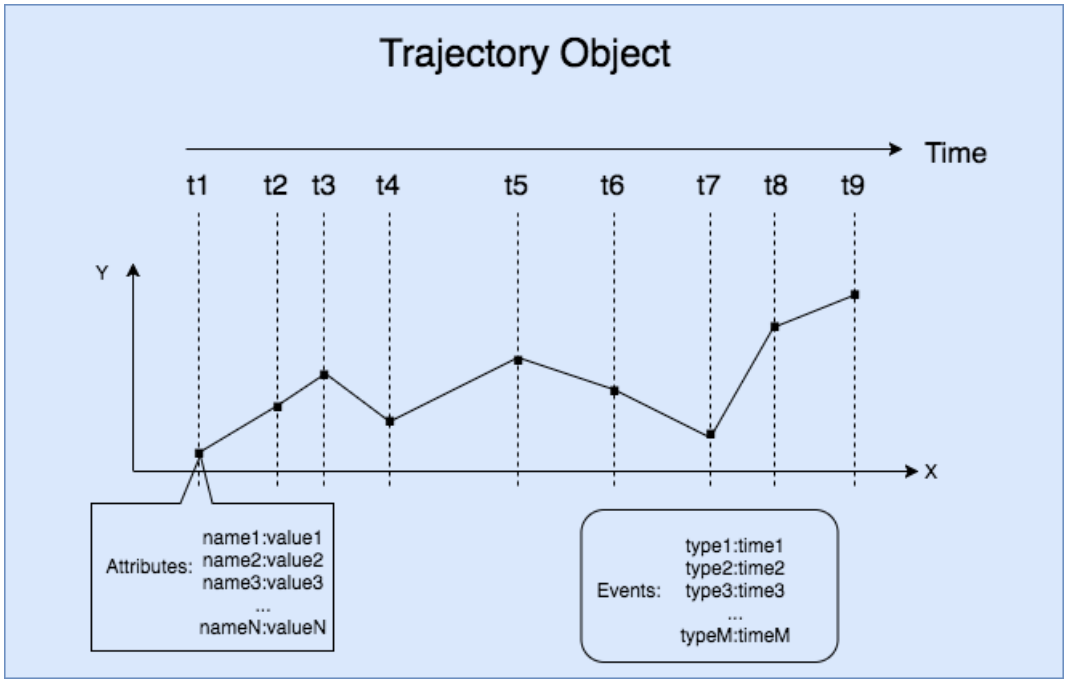
7.1. Terms

This topic introduces the basic concepts of Trajectory SQL.

Trajectory objects

Concept	Description
Trajectory Point	The spatio-temporal object that consists of the spatial location and attributes of a moving object at a specific point in time. The spatial location can be represented by 2D or 3D coordinates. The attributes can be a number of fields that use different data types.
Trajectory Object	The high-dimensional object that consists of trajectory points and events. Specifically, the high-dimensional object includes time, space, attributes, and events.
Trajectory Timeline	The time value sequence of a trajectory. The time values in the sequence must be consecutive.
Trajectory Spatial	The spatial geometry of a trajectory. In most cases, the spatial geometry is described by using the LineString data type.
Trajectory Leaf	The spatial location of a moving object at a specific point in time. The spatial location is a trajectory point.
Trajectory Attributes	The attributes that describe the trajectory of a moving object at different trajectory points. These attributes include velocity and direction.
Trajectory Attribute Field	The attribute field that describes the trajectory of a moving object. For example, these attribute fields include the velocity field. The number of values for an attribute field is the same as the number of trajectory points that is owned by the moving object.
Trajectory Field Value	The value of an attribute field at a specific point in time.
Trajectory Events	The events that occur along a trajectory. For example, these events include the refueling, breakdown, and car lock events of a moving car along the trajectory. An event consists of the event type ID and the event time.

The following figure shows the trajectory of a moving object.

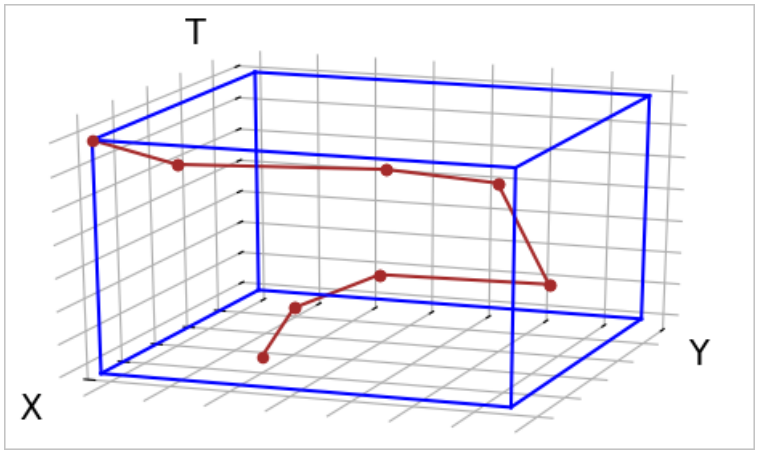


BoxNDF objects

Trajectory-related objects (trajectories and geometries) can be complex. Therefore, cubes that are easier to calculate than these trajectory-related objects may be used to describe queries or simplify computing operations. The BoxNDF data type is used to describe cubes.

A BoxNDF object is a multi-dimensional spatio-temporal cube that is expressed by the minimum and maximum values on the following four coordinate axes: x, y, z, and t. The z axis indicates space, and the t axis indicates time. Different cubes may include different axes. For example, some cubes include only the x and y axes, and some include all of the x, y, z, and t axes.

When you run a query, you can use a cube to specify the query scope. In addition, you can use a bounding box of the trajectory or geometry type to make the query easier. The following figure shows the trajectory and bounding box of an object that moves on the x, y, and t axes.



7.2. Constructors

7.2.1. Overview

Constructors include the function for constructing a trajectory object by using JSON-formatted strings or arrays and the function for appending trajectory points or a sub-trajectory to a trajectory object.

7.2.2. ST_append

This function appends trajectory points or a sub-trajectory to a trajectory object.

Syntax

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

Parameters

Parameter	Description
traj	The original trajectory object.
spatial	The spatial geometry object of the appended trajectory object.
timespan	The time array of the appended trajectory object. It can be a timeline.
str_attrs_json	The attributes of the appended trajectory object. For more information, see ST_makeTrajectory.
tail	The appended trajectory object.

Examples

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

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

7.2.3. ST_makeTrajectory

This function constructs a trajectory object.

Syntax

```
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);
```

Parameters

Parameter	Description
type	The trajectory type. Only ST_POINT is supported.

Parameter	Description
spatial	The spatial geometry object based on a linestring or point object.
timespan	The time range of the trajectory, which contains the start time and end time.
start	The start time of the trajectory.
end	The end time of the trajectory.
timeline	The trajectory timeline. The number of time points must be the same as that of points in the linestring object.
attrs_json	The trajectory attributes and events, in JSON format. The value can be null.
attr_field_names	The array of the names of all attribute fields for the trajectory.
x	The x-axis (array) for the geometry object.
y	The y-axis (array) for the geometry object.
srid	The spatial reference system identifier (SRID) of the trajectory. This parameter is required.

1. The format of the attrs_json parameter is as follows:

```
{
 "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 specifies the number of trajectory points in the trajectory object. Its value must be consistent with the number of spatial points in the spatial geometry object. Its value is also the same as the number of values for each attribute field. All attribute fields must have the same number of values.

attributes specifies trajectory attributes, including the definitions and a sequence of values for all attribute fields. It must co-exist with leafcount. Attribute definitions and requirements are as follows:

- An attribute name can contain a maximum of **60** characters.
- type: the field type. Valid values: integer, float, string, timestamp, and bool.
- length: the field length. The value varies with the field type. integer: 1, 2, 4, or 8. float: 4 or 8. string: customizable. (If not specified, the default length is 64 and the maximum length is 253. The length value is the actual number of characters and excludes the end identifier.) timestamp: If not specified, the default length is 8. bool: If not specified, the default length is 1.

- nullable: indicates whether the field can be null. Valid values: true and false. Default value: true.
- value: the sequence of field values, listed in JSON array format. If a value is null, null is displayed.

events specifies trajectory events. Multiple events are listed in JSON array format. Each array element is expressed in key:value format, where key indicates the event type and value indicates the event time.

2. If only the timespan parameter or the start and end parameters are specified as time, such as in syntax 1 and syntax 2, this constructor can interpolate time points based on the number of spatial points to generate the trajectory timeline.
3. If all the four types of syntax do not meet your requirements, you can customize the parameters following the first six fixed parameters to create a custom ST_makeTrajectory constructor as follows:

```
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;
```

Examples

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

```
----------------------------------------------------------------------------------------------------------------------------------------------------------------
 {"trajectory":{"version":1,"type":"STPOINT","leafcount":3,"start_time":"2010-01-01 14:30:00","end_time":
"2010-01-01 15:30:00","spatial":"SRID=4326;LINESTRING(114 35,115 36,116 37)","timeline":["2010-01-01 14:3
0:00","2010-01-01 15:00:00","2010-01-01 15:30:00"],"attributes":{"leafcount":3,"velocity":{"type":"integer","
length":2,"nullable":true,"value":[120,130,140]},"accuracy":{"type":"float","length":4,"nullable":false,"valu
e":[120.0,130.0,140.0]},"bearing":{"type":"float","length":8,"nullable":false,"value":[120.0,130.0,140.0]},"ves
name":{"type":"string","length":20,"nullable":true,"value":["adsf","sdf","sdfff"]},"active":{"type":"timesta
mp","length":8,"nullable":false,"value":["2010-01-01 14:30:00","2010-01-01 15:00:00","2010-01-01 15:30:00"]
}},"events":[{"1":"2010-01-01 14:30:00"},{"2":"2010-01-01 15:00:00"},{"3":"2010-01-01 15:30:00"}]}}
(1 row)
-- (3) ST_MakeTrajectory with a 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], '{"lea
fcount":3,"attributes":{"velocity": {"type": "integer", "length": 2,"nullable" : true,"value": [120, 130, 140]}, "a
ccuracy": {"type": "float", "length": 4, "nullable" : false,"value": [120, 130, 140]}, "bearing": {"type": "float", "l
ength": 8, "nullable" : false,"value": [120, 130, 140]}, "vesname": {"type": "string", "length": 20, "nullable" : tr
ue,"value": ["adsf", "sdf", "sdfff"]}, "active": {"type": "timestamp", "nullable" : false,"value": ["Fri Jan 01 14:3
0:00 2010", "Fri Jan 01 15:00:00 2010", "Fri Jan 01 15:30:00 2010"]}}, "events": [{"1" : "Fri Jan 01 14:30:00 2010
"}, {"2" : "Fri Jan 01 15:00:00 2010"}, {"3" : "Fri Jan 01 15:30:00 2010"}]}');                st_maketrajectory
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
 {"trajectory":{"version":1,"type":"STPOINT","leafcount":3,"start_time":"2010-01-01 14:30:00","end_time":
"2010-01-01 15:30:00","spatial":"SRID=4326;LINESTRING(114 35,115 36,116 37)","timeline":["2010-01-01 14:3
0:00","2010-01-01 15:00:00","2010-01-01 15:30:00"],"attributes":{"leafcount":3,"velocity":{"type":"integer","
length":2,"nullable":true,"value":[120,130,140]},"accuracy":{"type":"float","length":4,"nullable":false,"valu
e":[120.0,130.0,140.0]},"bearing":{"type":"float","length":8,"nullable":false,"value":[120.0,130.0,140.0]},"ves
name":{"type":"string","length":20,"nullable":true,"value":["adsf","sdf","sdfff"]},"active":{"type":"timesta
mp","length":8,"nullable":false,"value":["2010-01-01 14:30:00","2010-01-01 15:00:00","2010-01-01 15:30:00"]
}},"events":[{"1":"2010-01-01 14:30:00"},{"2":"2010-01-01 15:00:00"},{"3":"2010-01-01 15:30:00"}]}}
(1 row)
-- (4) ST_MakeTrajectory with null attrs_json
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","spati
al":"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 that constructs a trajectory object from points
select st_makeTrajectory('STPOINT'::leaftype, ARRAY[1::float8], ARRAY[2::float8], 4326, ARRAY['2010-01-01 1
1:30'::timestamp], ARRAY['velocity'], ARRAY[1::int4], NULL, NULL, NULL::anyarray);
                                    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":{"leaf
count":1,"velocity":{"type":"integer","length":4,"nullable":true,"value":[1]}}}}
(1 row)
```

7.3. Editing and processing functions

7.3.1. ST_attrDeduplicate

This function removes trajectory points whose values for the specified attribute field are duplicate from a trajectory object and returns the trajectory object after deduplication. The start and end trajectory points of the trajectory object must be kept.

Syntax

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

Parameters

Parameter	Description
traj	The trajectory object.
attr_field_name	The name of the specified attribute field.

Examples

```
select st_attrDeduplicate(ST_makeTrajectory('STPOINT'::leaftype, 'LINESTRING(-179.48077 51.72814,-179.46731 51.74634,-179.46502 51.74934,-179.46183 51.75378,-179.45943 51.75736,-179.45560 51.76273,-179.44845 51.77186,-179.43419 51.78977,-179.41259 51.81643,-179.41001 51.81941,-179.40751 51.82223,-179.40497 51.82505,-179.40242 51.82796,-179.39981 51.83095,-179.39734 51.83398,-179.39499 51.83709)'::geometry, ARRAY['2017-01-15 09:06:39'::timestamp,'2017-01-15 09:13:39','2017-01-15 09:14:48','2017-01-15 09:16:28','2017-01-15 09:17:48','2017-01-15 09:19:48','2017-01-15 09:23:19','2017-01-15 09:30:28','2017-01-15 09:34:40','2017-01-15 09:36:59','2017-01-15 09:38:09','2017-01-15 09:39:18','2017-01-15 09:40:40','2017-01-15 09:47:38','2017-01-15 09:48:49','2017-01-15 21:18:30'], '{"leafcount": 16, "attributes": {"heading": {"type": "float", "length": 4, "nullable": false,"value": [23.0,23.0,23.0,23.0,21.0,21.0,72.0,72.0,72.0,72.0,73.0,74.0,73.0,73.0,73.0,73.0]}}}'),'heading') as traj;
                                   traj
----------------------------------------------------------------------------------------------------------------------------------------------------------------------
 {"trajectory":{"version":1,"type":"STPOINT","leafcount":6,"start_time":"2017-01-15 09:17:48","end_time":"2017-01-15 21:18:30","spatial":"LINESTRING(-179.45943 51.75736,-179.44845 51.77186,-179.40751 51.82223,-179.40497 51.82505,-179.39499 51.83709)","timeline":["2017-01-15 09:17:48","2017-01-15 09:23:19","2017-01-15 09:38:09","2017-01-15 09:39:18","2017-01-15 21:18:30"],"attributes":{"leafcount":6,"heading":{"type":"float","length":4,"nullable":false,"value":[23.0,21.0,72.0,73.0,74.0,73.0]}}}}
(1 row)
```

7.3.2. ST_Compress

This function compresses a trajectory object based on specified thresholds.

Syntax

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

Parameters

Parameter	Description
traj	The original trajectory object.
dist	The Euclidean distance offset threshold. After this value is specified, trajectory points whose Euclidean distance offset is greater than this value are kept, so as to maintain the spatial trend of the original trajectory object.
angle	The angle offset threshold. After this value is specified, trajectory points whose angle changes are greater than this value are kept.
acceleration	The acceleration threshold. After this value is specified, trajectory points whose velocity changes are greater than this value are kept.
velocity_field	The name of the velocity attribute field in the trajectory object. The values of this attribute field are used to compute the acceleration.

Description

This function discards trajectory points based on specified thresholds to compress the trajectory object in lossy mode, and returns the compressed trajectory object.

- Syntax 1: compresses the trajectory object based on the specified offset threshold for the spatial distance. This function maintains the spatial trend of the original trajectory object, but may delete important trajectory points with large angle or velocity changes.
- Syntax 2: compresses the trajectory object based on the specified offset threshold for the spatial distance, angle offset threshold, and acceleration threshold. This function not only maintains the spatial trend of the original trajectory object, but also keeps important trajectory points with large angle or velocity changes. You can specify any one or two of the thresholds for compression.
- Syntax 3: achieves the same effect as syntax 2. This syntax applies when the trajectory object contains the velocity attribute field. After the name of the velocity attribute field is specified, this function directly computes the acceleration of trajectory points based on the values of the velocity attribute field.

Examples

```
-- Create data.
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,-179.4341
9 51.78977,-179.42595 51.80094,-179.42343 51.80411,-179.42078 51.80719,-179.41821 51.81025,-179.41562 51.
81308,-179.41259 51.81643,-179.41001 51.81941,-179.40751 51.82223,-179.40497 51.82505,-179.40242 51.8279
6,-179.39981 51.83095,-179.39734 51.83398,-179.39499 51.83709,-179.39264 51.84023,-179.39037 51.84333,-17
9.38699 51.84791,-179.38467 51.85114,-179.38216 51.85439,-179.37997 51.85762,-179.37772 51.86144,-179.37
474 51.86568,-179.37219 51.86869,-179.36983 51.87156,-179.36755 51.87467,-179.36001 51.88423,-179.35754
51.88712,-179.34216 51.90644,-179.33935 51.90995,-179.33704 51.91298,-179.18826 52.10105,-179.18096 52.1
```

```
1031,-179.17504 52.11786,-179.16482 52.12996,-179.16233 52.13289,-179.15967 52.13590,-179.14599 52.15132
,-177.76666 52.85042,-177.48459 52.89898,-177.47841 52.90001,-177.47319 52.90084,-177.46251 52.90268,-177
.38188 52.91595,-177.37102 52.91765,-177.36378 52.91877,-177.34492 52.92173,-177.33217 52.92364,-177.325
81 52.92468,-177.31238 52.92697,-177.03751 52.97394,-176.93063 52.99160,-176.92406 52.99265,-176.91471 5
2.99423,-176.90643 52.99554,-176.89912 52.99674,-176.89246 52.99791,-176.88342 52.99942,-176.87697 53.00
060,-176.86594 53.00256,-176.85946 53.00370,-176.85294 53.00481,-176.84640 53.00592,-176.83985 53.00705,-
176.83238 53.00830,-176.82589 53.00950,-176.81848 53.01084,-176.80553 53.01310,-176.79879 53.01419,-176.
79115 53.01548,-176.78466 53.01668,-176.77901 53.01765,-176.77256 53.01879,-176.76301 53.02039,-176.7564
9 53.02141,-176.74700 53.02296,-176.73757 53.02450,-176.71683 53.02795,-176.70741 53.02950,-176.68481 53.
03327)'::geometry, ARRAY['2017-01-15 09:06:39'::timestamp,'2017-01-15 09:10:08'::timestamp,'2017-01-15 09
:11:20'::timestamp,'2017-01-15 09:12:29'::timestamp,'2017-01-15 09:13:39'::timestamp,'2017-01-15 09:14:48'::
timestamp,'2017-01-15 09:16:28'::timestamp,'2017-01-15 09:17:48'::timestamp,'2017-01-15 09:19:48'::timesta
mp,'2017-01-15 09:23:19'::timestamp,'2017-01-15 09:30:28'::timestamp,'2017-01-15 09:34:40'::timestamp,'20
17-01-15 09:35:49'::timestamp,'2017-01-15 09:36:59'::timestamp,'2017-01-15 09:38:09'::timestamp,'2017-01-1
5 09:39:18'::timestamp,'2017-01-15 09:40:40'::timestamp,'2017-01-15 09:41:49'::timestamp,'2017-01-15 09:42:
58'::timestamp,'2017-01-15 09:44:08'::timestamp,'2017-01-15 09:45:18'::timestamp,'2017-01-15 09:46:29'::tim
estamp,'2017-01-15 09:47:38'::timestamp,'2017-01-15 09:48:49'::timestamp,'2017-01-15 09:49:58'::timestamp
,'2017-01-15 09:51:08'::timestamp,'2017-01-15 09:52:49'::timestamp,'2017-01-15 09:53:58'::timestamp,'2017-
01-15 09:55:09'::timestamp,'2017-01-15 09:56:18'::timestamp,'2017-01-15 09:57:38'::timestamp,'2017-01-15 0
9:59:09'::timestamp,'2017-01-15 10:00:20'::timestamp,'2017-01-15 10:01:29'::timestamp,'2017-01-15 10:02:39'
::timestamp,'2017-01-15 10:06:29'::timestamp,'2017-01-15 10:07:40'::timestamp,'2017-01-15 10:15:00'::timest
amp,'2017-01-15 10:16:20'::timestamp,'2017-01-15 10:17:29'::timestamp,'2017-01-15 11:30:09'::timestamp,'2
017-01-15 11:33:58'::timestamp,'2017-01-15 11:36:58'::timestamp,'2017-01-15 11:42:00'::timestamp,'2017-01-
15 11:43:10'::timestamp,'2017-01-15 11:44:20'::timestamp,'2017-01-15 11:50:28'::timestamp,'2017-01-15 18:0
1:00'::timestamp,'2017-01-15 18:54:13'::timestamp,'2017-01-15 18:55:21'::timestamp,'2017-01-15 18:56:22'::ti
mestamp,'2017-01-15 18:58:21'::timestamp,'2017-01-15 19:13:21'::timestamp,'2017-01-15 19:15:21'::timesta
mp,'2017-01-15 19:16:41'::timestamp,'2017-01-15 19:20:11'::timestamp,'2017-01-15 19:22:31'::timestamp,'20
17-01-15 19:23:41'::timestamp,'2017-01-15 19:26:10'::timestamp,'2017-01-15 20:15:49'::timestamp,'2017-01-1
5 20:34:39'::timestamp,'2017-01-15 20:35:49'::timestamp,'2017-01-15 20:37:30'::timestamp,'2017-01-15 20:39:
00'::timestamp,'2017-01-15 20:40:19'::timestamp,'2017-01-15 20:41:30'::timestamp,'2017-01-15 20:43:08'::tim
estamp,'2017-01-15 20:44:19'::timestamp,'2017-01-15 20:46:19'::timestamp,'2017-01-15 20:47:29'::timestamp
,'2017-01-15 20:48:40'::timestamp,'2017-01-15 20:49:49'::timestamp,'2017-01-15 20:50:59'::timestamp,'2017-
01-15 20:52:21'::timestamp,'2017-01-15 20:53:29'::timestamp,'2017-01-15 20:54:50'::timestamp,'2017-01-15 2
0:57:09'::timestamp,'2017-01-15 20:58:20'::timestamp,'2017-01-15 20:59:40'::timestamp,'2017-01-15 21:00:49'
::timestamp,'2017-01-15 21:01:50'::timestamp,'2017-01-15 21:02:58'::timestamp,'2017-01-15 21:04:40'::timest
amp,'2017-01-15 21:05:50'::timestamp,'2017-01-15 21:07:29'::timestamp,'2017-01-15 21:09:11'::timestamp,'2
017-01-15 21:12:49'::timestamp,'2017-01-15 21:14:30'::timestamp,'2017-01-15 21:18:30'::timestamp], '{"leafc
ount": 89,"attributes" : {"sog" : {"type":"float","length":8,"nullable":false,"value":[10.5,10.4,10.5,10.7,10.8,1
0.3,10.7,10.4,10.5,10.1,10.2,11.0,11.2,10.8,10.3,10.3,10.1,10.7,10.6,10.0,10.3,10.5,10.6,10.3,10.8,10.9,10.8,10.8,
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,"nul
lable":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
.4,28.2,30.4,29.4,29.2,23.0,25.1,25.1,23.5,22.7,27.1,23.3,19.2,27.1,31.0,28.8,22.0,30.1,24.6,26.2,26.7,24.7,26.8,2
9.5,19.9,30.1,28.8,28.7,30.0,74.2,69.1,75.2,81.3,81.3,80.1,72.6,82.4,74.2,74.9,67.7,73.4,74.2,72.2,80.5,78.6,77.3,
70.9,80.1,85.4,71.9,67.0,77.5,77.5,72.2,70.5,72.6,70.8,77.8,71.2,71.2,73.8,75.4,67.1,77.5,74.3,76.9,80.1,72.8,76.0
,75.4,72.9]},"heading" : {"type":"float","length":8,"nullable":false,"value":[22.0,23.0,23.0,23.0,23.0,21.0,21.0,
25.0,24.0,26.0,25.0,27.0,28.0,29.0,31.0,30.0,28.0,29.0,29.0,28.0,28.0,27.0,24.0,24.0,25.0,25.0,25.0,26.0,25.0,24.0
,25.0,29.0,31.0,28.0,29.0,31.0,28.0,29.0,29.0,29.0,27.0,27.0,26.0,26.0,26.0,27.0,27.0,69.0,71.0,72.0,72.0,71.0,73.
0,72.0,72.0,72.0,72.0,71.0,71.0,72.0,72.0,72.0,73.0,72.0,71.0,72.0,72.0,72.0,71.0,72.0,72.0,73.0,71.0,72.0,71.0,72
.0,73.0,72.0,72.0,72.0,72.0,73.0,74.0,73.0,73.0,73.0,73.0,73.0,73.0]}}}'));
-- Compress the trajectory object.
select st_compress(traj,0.001) as traj from traj_test;
        trai
```

```
-----------------------------------------------------------------
 {"trajectory":{"version":1,"type":"STPOINT","leafcount":8,"start_time":"2017-01-15 09:06:39","end_time":
"2017-01-15 21:18:30","spatial":"LINESTRING(-179.48077 51.72814,-179.42595 51.80094,-179.39734 51.83398,-
179.37474 51.86568,-179.17504 52.11786,-179.14599 52.15132,-177.76666 52.85042,-176.68481 53.03327)","ti
meline":["2017-01-15 09:06:39","2017-01-15 09:34:40","2017-01-15 09:47:38","2017-01-15 09:59:09","2017-01-
15 11:36:58","2017-01-15 11:50:28","2017-01-15 18:01:00","2017-01-15 21:18:30"],"attributes":{"leafcount":8,
"sog":{"type":"float","length":8,"nullable":false,"value":[10.5,11.0,10.6,11.2,10.1,9.9,12.3,12.7]},"cog":{"type
":"float","length":8,"nullable":false,"value":[23.3,28.5,29.2,27.1,19.9,30.0,74.2,72.9]},"heading":{"type":"floa
t","length":8,"nullable":false,"value":[22.0,27.0,24.0,29.0,26.0,27.0,69.0,73.0]}}}}
(1 row)
select st_compress(traj,0.001,null,null) as traj from traj_test where traj_id=5;
        traj
-----------------------------------------------------------------
{"trajectory":{"type":"STPOINT","leafsize":8,"starttime":"2017-01-15 09:06:39","endtime":"2017-01-15 21:18
:30","spatial":"LINESTRING(-179.48077 51.72814,-179.42595 51.80094,-179.39734 51.83398,-179.37474 51.865
68,-179.17504 52.11786,-179.14599 52.15132,-177.76666 52.85042,-176.68481 53.03327)","timeline":["2017-01
-15 09:06:39","2017-01-15 09:34:40","2017-01-15 09:47:38","2017-01-15 09:59:09","2017-01-15 11:36:58","201
7-01-15 11:50:28","2017-01-15 18:01:00","2017-01-15 21:18:30"],"themeline":{"leafs":8,"sog":[10.5,11.0,10.6,1
1.2,10.1,9.9,12.3,12.7],"cog":[23.3,28.5,29.2,27.1,19.9,30.0,74.2,72.9],"heading":[22.0,27.0,24.0,29.0,26.0,27.0,
69.0,73.0]}}}
(1 row)
select st_compress(traj,0.001,5,0.3) as traj from traj_test;
        traj
-----------------------------------------------------------------
{"trajectory":{"version":1,"type":"STPOINT","leafcount":13,"start_time":"2017-01-15 09:06:39","end_time":
"2017-01-15 21:18:30","spatial":"LINESTRING(-179.48077 51.72814,-179.42595 51.80094,-179.39734 51.83398,-
179.37474 51.86568,-179.35754 51.88712,-179.18826 52.10105,-179.17504 52.11786,-179.14599 52.15132,-177.
76666 52.85042,-177.47841 52.90001,-177.47319 52.90084,-176.83238 53.0083,-176.68481 53.03327)","timelin
e":["2017-01-15 09:06:39","2017-01-15 09:34:40","2017-01-15 09:47:38","2017-01-15 09:59:09","2017-01-15 10:
07:40","2017-01-15 11:30:09","2017-01-15 11:36:58","2017-01-15 11:50:28","2017-01-15 18:01:00","2017-01-15
18:55:21","2017-01-15 18:56:22","2017-01-15 20:52:21","2017-01-15 21:18:30"],"attributes":{"leafcount":13,"
sog":{"type":"float","length":8,"nullable":false,"value":[10.5,11.0,10.6,11.2,10.4,9.1,10.1,9.9,12.3,12.0,12.0,1
2.5,12.7]},"cog":{"type":"float","length":8,"nullable":false,"value":[23.3,28.5,29.2,27.1,24.6,26.8,19.9,30.0,74.
2,75.2,81.3,72.6,72.9]},"heading":{"type":"float","length":8,"nullable":false,"value":[22.0,27.0,24.0,29.0,28.0,
27.0,26.0,27.0,69.0,72.0,72.0,72.0,73.0]}}}}
(1 row)
select st_compress(traj,0.001,5,1.1,'sog') as traj from traj_test;
        traj
-----------------------------------------------------------------
{"trajectory":{"version":1,"type":"STPOINT","leafcount":10,"start_time":"2017-01-15 09:06:39","end_time":
"2017-01-15 21:18:30","spatial":"LINESTRING(-179.48077 51.72814,-179.42595 51.80094,-179.39734 51.83398,-
179.37474 51.86568,-179.33704 51.91298,-179.18826 52.10105,-179.17504 52.11786,-179.14599 52.15132,-177.
76666 52.85042,-176.68481 53.03327)","timeline":["2017-01-15 09:06:39","2017-01-15 09:34:40","2017-01-15 0
9:47:38","2017-01-15 09:59:09","2017-01-15 10:17:29","2017-01-15 11:30:09","2017-01-15 11:36:58","2017-01-
15 11:50:28","2017-01-15 18:01:00","2017-01-15 21:18:30"],"attributes":{"leafcount":10,"sog":{"type":"float"
,"length":8,"nullable":false,"value":[10.5,11.0,10.6,11.2,10.6,9.1,10.1,9.9,12.3,12.7]},"cog":{"type":"float","le
ngth":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)
```

7.3.3. ST_CompressSED

This function compresses a trajectory object based on the offset threshold of the synchronous Euclidean distance (SED).

Syntax

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

Parameters

Parameter	Description
traj	The original trajectory object.
dist	The SED offset threshold. After this value is specified, trajectory points whose SED offset is greater than this value are kept, so as to maintain the spatial trend of the original trajectory object.

Description

This function calculates the SED of trajectory points, discards the points whose SED offset is smaller than the SED offset threshold to compress the trajectory object in lossy mode, and then returns the compressed trajectory object.

Examples

```
select st_compressSED(traj, 0.001) as traj from traj_test;
        traj
-------------------------------------------------------------
{"trajectory":{"version":1,"type":"STPOINT","leafcount":12,"start_time":"2017-01-15 09:06:39","end_time":"2017-01-15 21:18:30","spatial":"LINESTRING(-179.48077 51.72814,-179.42595 51.80094,-179.39734 51.83398,-179.37474 51.86568,-179.18826 52.10105,-179.16482 52.12996,-179.14599 52.15132,-177.76666 52.85042,-177.47319 52.90084,-177.31238 52.92697,-177.03751 52.97394,-176.68481 53.03327)","timeline":["2017-01-15 09:06:39","2017-01-15 09:34:40","2017-01-15 09:47:38","2017-01-15 09:59:09","2017-01-15 11:30:09","2017-01-15 11:42:00","2017-01-15 11:50:28","2017-01-15 18:01:00","2017-01-15 18:56:22","2017-01-15 19:26:10","2017-01-15 20:15:49","2017-01-15 21:18:30"],"attributes":{"leafcount":12,"sog":{"type":"float","length":8,"nullable":false,"value":[10.5,11.0,10.6,11.2,9.1,9.7,9.9,12.3,12.0,12.2,12.8,12.7]},"cog":{"type":"float","length":8,"nullable":false,"value":[23.3,28.5,29.2,27.1,26.8,30.1,30.0,74.2,81.3,73.4,74.2,72.9]},"heading":{"type":"float","length":8,"nullable":false,"value":[22.0,27.0,24.0,29.0,27.0,26.0,27.0,69.0,72.0,71.0,72.0,73.0]}}}}
(1 row)
```

7.3.4. ST_deviation

This function returns the deviation of the processed trajectory object from the original trajectory object.

Syntax

```
float8 ST_deviation(trajectory traj, trajectory after_oper_traj);
```

Parameters

Parameter	Description
traj	The original trajectory object.
after_oper_traj	The processed trajectory object, such as the compressed trajectory object.

Examples

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

7.3.5. ST_sort

This function sorts the timeline of a trajectory object in ascending order and returns the trajectory object after sorting.

Syntax

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

Parameters

Parameter	Description
traj	The trajectory object.

Examples

```
select st_sort(ST_makeTrajectory('STPOINT'::leaftype, 'LINESTRING(-179.48077 51.72814,-179.46731 51.74634,-179.46502 51.74934,-179.46183 51.75378,-179.45943 51.75736,-179.45560 51.76273,-179.44845 51.77186,-179.43419 51.78977,-179.41259 51.81643,-179.41001 51.81941,-179.40751 51.82223,-179.40497 51.82505,-179.40242 51.82796,-179.39981 51.83095,-179.39734 51.83398,-179.39499 51.83709)'::geometry, ARRAY['2017-01-15 09:06:39'::timestamp,'2017-01-15 09:14:48'::timestamp,'2017-01-15 09:13:39'::timestamp,'2017-01-15 09:16:28'::timestamp,'2017-01-15 09:19:48'::timestamp,'2017-01-15 09:17:48'::timestamp,'2017-01-15 09:23:19'::timestamp,'2017-01-15 09:34:40'::timestamp,'2017-01-15 09:30:28'::timestamp,'2017-01-15 09:36:59'::timestamp,'2017-01-15 09:38:09'::timestamp,'2017-01-15 09:39:18'::timestamp,'2017-01-15 09:40:40'::timestamp,'2017-01-15 09:47:38'::timestamp,'2017-01-15 21:18:30'::timestamp,'2017-01-15 09:48:49'::timestamp], '{"leafcount": 16, "attributes": {"heading": {"type": "integer", "length": 4, "nullable": false,"value":[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]}}}'));
                                st_sort
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
 {"trajectory":{"version":1,"type":"STPOINT","leafcount":16,"start_time":"2017-01-15 09:06:39","end_time":"2017-01-15 21:18:30","spatial":"LINESTRING(-179.48077 51.72814,-179.46502 51.74934,-179.46731 51.74634,-179.46183 51.75378,-179.4556 51.76273,-179.45943 51.75736,-179.44845 51.77186,-179.41259 51.81643,-179.43419 51.78977,-179.41001 51.81941,-179.40751 51.82223,-179.40497 51.82505,-179.40242 51.82796,-179.39981 51.83095,-179.39499 51.83709,-179.39734 51.83398)","timeline":["2017-01-15 09:06:39","2017-01-15 09:13:39","2017-01-15 09:14:48","2017-01-15 09:16:28","2017-01-15 09:17:48","2017-01-15 09:19:48","2017-01-15 09:23:19","2017-01-15 09:30:28","2017-01-15 09:34:40","2017-01-15 09:36:59","2017-01-15 09:38:09","2017-01-15 09:39:18","2017-01-15 09:40:40","2017-01-15 09:47:38","2017-01-15 09:48:49","2017-01-15 21:18:30"],"attributes":{"leafcount":16,"heading":{"type":"integer","length":4,"nullable":false,"value":[0,2,1,3,5,4,6,8,7,9,10,11,12,13,15,14]}}}}
(1 row)
```

7.3.6. ST_removeDriftPoints

You can use this function to delete trajectory points that meet the specified threshold conditions from a trajectory object.

Syntax

```
trajectory ST_removeDriftPoints(trajectory traj, float8 speed, float8 dist, interval offset);
```

Parameters

Parameter	Description
traj	The trajectory object.
speed	The speed threshold. The unit is meter per second.
dist	The distance threshold. The unit is meter.
offset	The time interval threshold.

Description

This function deletes the trajectory points that meet the specified conditions from a trajectory object and returns the new data of the trajectory object. The returned trajectory object contains at least two trajectory points. This function deletes a trajectory point if the trajectory point meets the following three conditions: The speed of the trajectory point is greater than the specified speed threshold, the distance from the trajectory point to the previous trajectory point is greater than the specified distance threshold, and the time interval is greater than the specified offset threshold. The unit of speed is meter per second and the unit of distance is meter. If the specified trajectory object has only two trajectory points, no trajectory point is deleted.

The speed parameter specifies a speed threshold. The speed threshold is calculated in real time based on the position of the trajectory point and the time offset. The unit of speed is meter per second. The dist parameter specifies a distance threshold. The unit of distance is meter. The offset parameter specifies a time interval. By default, the coordinate reference system of a trajectory object is World Geodetic System 1984 (WGS 84). If you do not want to use WGS 84, you can specify a spatial reference system identifier (SRID) when you create a trajectory object or invoke the ST_SetSRID function to set an SRID for the trajectory object. The system calculates the spherical distance between two points based on the specified SRID. The unit of spherical distances is meter.

Examples

```
select id, st_leafcount(traj) from table_name where id = 1;
 id | st_leafcount
----+--------------
  1 |        53
(1 row)
update table_name set traj=st_removeDriftPoints(traj,25.72, 9620, interval '30 seconds') where id = 1;
--Two trajectory points are deleted.
select id, st_leafcount(traj) from table_name where id = 1;
 id | st_leafcount
----+--------------
  1 |        49
(1 row)
```

7.3.7. ST_Split

The ST_Split function is used to split a trajectory into multiple sub-trajectories based on a spatial geometry object.

Syntax

```
trajectory[] ST_Split(trajectory traj, geometry geom, float8 radius_of_buffer);
trajectory[] ST_Split(trajectory traj, text config);
```

Parameters

Parameter	Description
traj	The trajectory that you want to split.
geom	The spatial geometry object that is used to split the trajectory. Only Point and MultiPoint geometry objects are supported.

Parameter	Description
radius_of_buffer	The radius of the buffer that is built based on the Point geometry object. Unit: meter.
config	The rule that specifies how to split the trajectory.

Description

- Configuration 1: `ST_Split(trajectory traj, geometry geom, float8 radius_of_buffer)`

  This function splits a trajectory based on the specified geometry object and returns an array of sub-trajectories. If the trajectory is split based on a MultiPoint spatial geometry object, the trajectory may be split into multiple segments.

- Configuration 2: `ST_Split(trajectory traj, text config)`

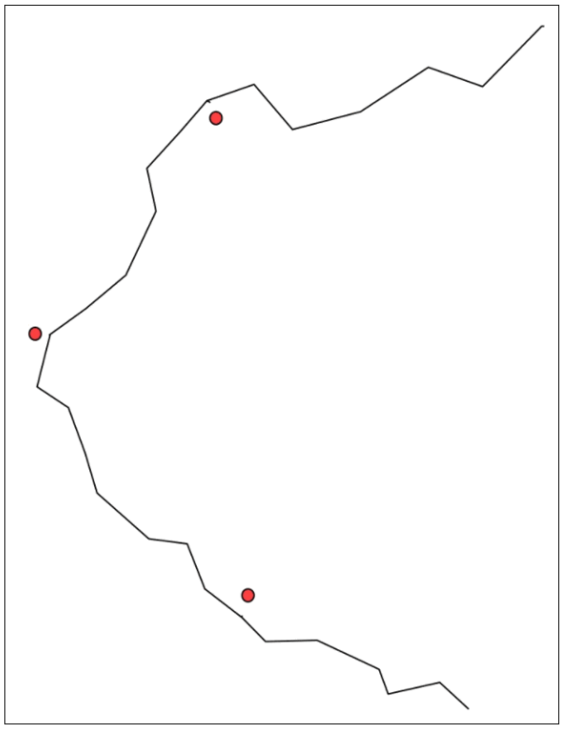
  This function splits a trajectory based on the specified rule and returns an array of sub-trajectories. Only one rule can be specified by the config parameter. The following table describes the rules that you can specify by using the config parameter.

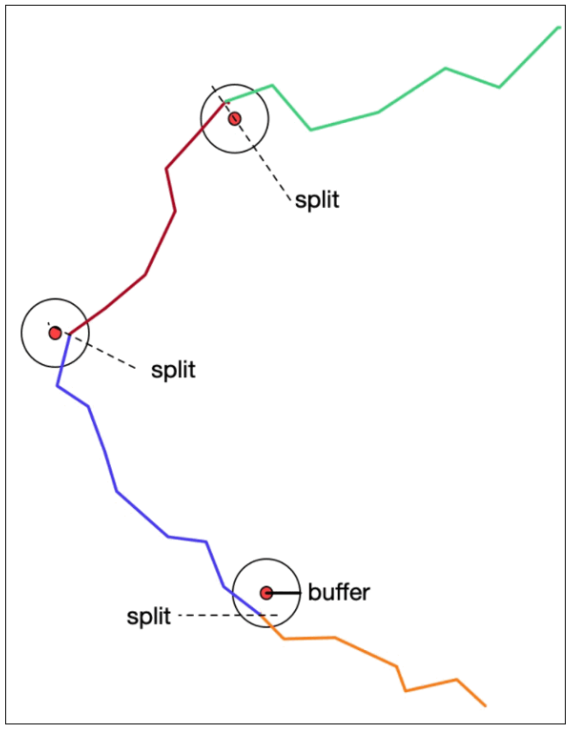
Rule name	Type	Description
cut_point.max_point	Positive integer	This rule specifies a quantity interval between sampling points. The function splits the trajectory into sub-trajectories based on the quantity interval that you specify.
cut_point.even_divide	Positive integer	This rule specifies the number of parts into which the trajectory is divided. The function evenly splits the trajectory into the specified number of sub-trajectories. If the number of edges of the trajectory is smaller than the value that is specified by the rule, the function considers each edge of the trajectory as a sub-trajectory.
cut_edge.time_interval	Positive time range	This rule specifies a time interval. The function splits the trajectory into sub-trajectories based on the time interval that you specify.
cut_edge.geohash	Positive even number	This rule specifies a Geohash. The function splits the trajectory based on the Geohash that you specify in the rule and ensures that each sub-trajectory is included in one Geohash grid. If you specify this rule, make sure that the data of the trajectory is represented by a latitude and a longitude.

Rule name	Type	Description
drop_edge.temporal_length	Time range	This rule specifies a time interval. The function deletes the part between the specified time interval from the trajectory.
drop_edge.spatial_distance_2d	Floating point	This rule specifies a spatial distance. The function deletes the part specified by the spatial distance from the trajectory. The spatial distance is a two-dimensional Euclidean distance.

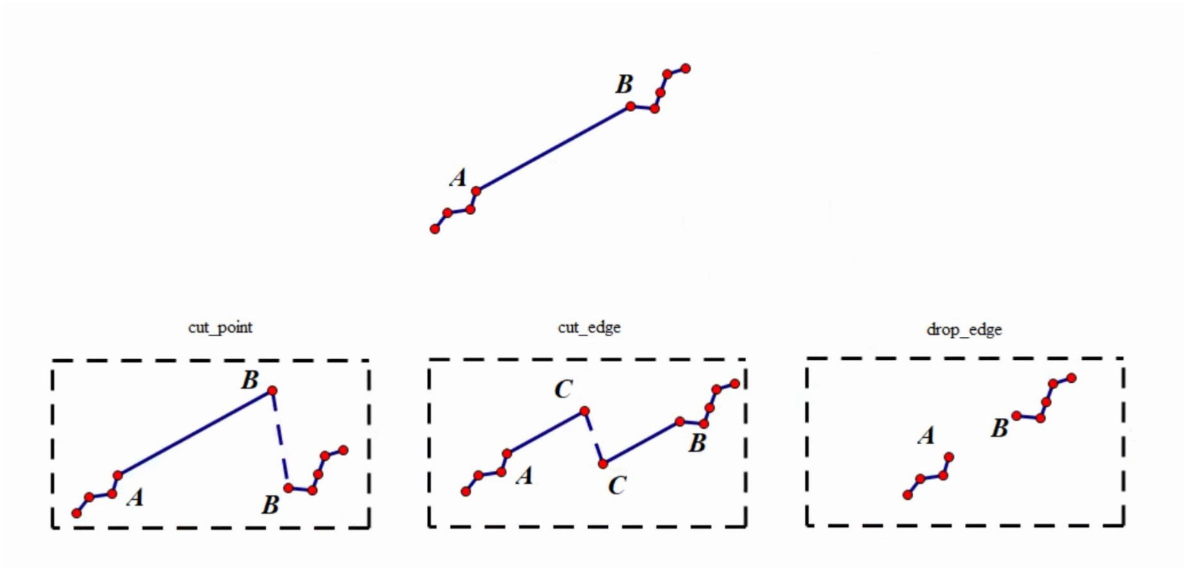
Diagrams

- Split a trajectory based on a specified geometry object.





- If you specify the cut_point.max_point or the cut_point.even_divide rule in the config parameter, the function splits the trajectory based on the specified sampling points. As shown in the following diagram, the function splits the trajectory into two sub-trajectories at Point B. If you specify the cut_edge.time_interval or the cut_edge.geohash rule in the config parameter, the function splits the trajectory based on the specified edges. As shown in the following diagram, the function splits the trajectory into two sub-trajectories at Point C. If you specify the drop_edge.temporal_length or the drop_edge.spatial_distance_2d rule in the config parameter, the function deletes the specified edge between the specified sampling points. As shown in the following diagram, the function deletes the edge between Point A and Point B.



Example

```
create table tr_split_traj(id integer, traj trajectory);
INSERT INTO tr_split_traj VALUES(3, ST_MakeTrajectory('STPOINT'::leaftype, st_geomfromtext('LINESTRING
(99.027 29.7555,99.313 29.9975,99.852 30.0745,104.879 35.0795,105.044 35.1235,105.187 35.0685,109.906 35.0
795,110.071 35.1675,110.192 35.0355,110.544 35.0245,111.017 34.8045)', 4326), ARRAY['2010-01-01 14:30'::tim
estamp,'2010-01-01 15:00','2010-01-01 15:10','2010-01-01 15:20','2010-01-01 15:30','2010-01-01 15:40','2010-0
1-01 15:50','2010-01-01 16:00','2010-01-01 16:10','2010-01-01 16:20','2010-01-01 16:30'],'{"leafcount":11,"attri
butes":{"velocity": {"type": "integer", "length": 2,"nullable" : true,"value": [120, 130, 140, 150, 160, 170, 180,
190, 200, 210, 220]}}}'));
select id, unnest(st_split(traj, st_geomfromtext('MULTIPOINT(100 30,105 35,110 35)'), 23000)) as subtraj fro
m tr_split_traj;
 id |                                                                                                    su
btraj
----+-----------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------
-----------------------------------
  3 | {"trajectory":{"version":1,"type":"STPOINT","leafcount":3,"start_time":"Fri Jan 01 14:30:00 2010","end_
time":"Fri Jan 01 15:10:00 2010","spatial":"SRID=4326;LINESTRING(99.027 29.7555,99.313 29.9975,99.852 30.0
745)","timeline":["Fri Jan 01 14:30:00 2010","Fri Jan 01 15:00:00 2010","Fri Jan 01 15:10:00 2010"],"attributes
":{"leafcount":3,"velocity":{"type":"integer","length":2,"nullable":true,"value":[120,130,140]}}}}
  3 | {"trajectory":{"version":1,"type":"STPOINT","leafcount":2,"start_time":"Fri Jan 01 15:10:00 2010","end_
time":"Fri Jan 01 15:20:00 2010","spatial":"SRID=4326;LINESTRING(99.852 30.0745,104.879 35.0795)","timelin
e":["Fri Jan 01 15:10:00 2010","Fri Jan 01 15:20:00 2010"],"attributes":{"leafcount":2,"velocity":{"type":"inte
ger","length":2,"nullable":true,"value":[140,150]}}}}
  3 | {"trajectory":{"version":1,"type":"STPOINT","leafcount":2,"start_time":"Fri Jan 01 15:40:00 2010","end_
time":"Fri Jan 01 15:50:00 2010","spatial":"SRID=4326;LINESTRING(105.187 35.0685,109.906 35.0795)","timeli
ne":["Fri Jan 01 15:40:00 2010","Fri Jan 01 15:50:00 2010"],"attributes":{"leafcount":2,"velocity":{"type":"int
eger","length":2,"nullable":true,"value":[170,180]}}}}
  3 | {"trajectory":{"version":1,"type":"STPOINT","leafcount":3,"start_time":"Fri Jan 01 16:10:00 2010","end_
time":"Fri Jan 01 16:30:00 2010","spatial":"SRID=4326;LINESTRING(110.192 35.0355,110.544 35.0245,111.017
34.8045)","timeline":["Fri Jan 01 16:10:00 2010","Fri Jan 01 16:20:00 2010","Fri Jan 01 16:30:00 2010"],"attrib
utes":{"leafcount":3,"velocity":{"type":"integer","length":2,"nullable":true,"value":[200,210,220]}}}}
With traj as(
  select
```

'{"trajectory":{"version":1,"type":"STPOINT","leafcount":19,"start_time":"2000-01-01 00:01:19.067179",
"end_time":"2000-01-01 03:24:25.946085","spatial":"LINESTRING(-100 -100 -100,-88.8925775739675 -86.6512
698383691 -92.3767832526937,-79.6904716538265 -80.6515727923252 -84.2357598245144,-75.843550771164
4 -73.7572890928326 -80.5007370118983,-70.6238425321256 -67.8213750167439 -74.5733173238113,-61.6014
582272619 -61.0636760429479 -67.9874239303172,-56.1098577060426 -54.4264591250879 -64.5007972046733
,-46.9800617334743 -49.4026757289345 -61.6160059720278,-41.7122942996211 -46.3224360072054 -56.52831
47455193,-35.5646221285375 -38.1688933617746 -49.2775720101781,-31.7230528349367 -33.6970051738123
-44.1693710885011,-23.1585765127093 -26.5895827477798 -40.6539742602035,-16.7020264320696 -21.61338
77349397 -37.3055470525287,-12.1044529232507 -14.1236051704424 -28.2295028120279,-3.77185660181567
-7.74744770256802 -24.3842111621052,0.488159407706304 -3.68223926316326 -19.9478872027248,6.3340688
1305078 4.54123636645575 -15.0410129944794,15.6666049417108 10.5611746329814 -11.2770220567472,14 1
1 -10)","timeline":["2000-01-01 00:01:19.067179","2000-01-01 00:12:36.116007","2000-01-01 00:23:53.164835"
,"2000-01-01 00:35:10.213663","2000-01-01 00:46:27.262491","2000-01-01 00:57:44.311319","2000-01-01 01:09
:01.360147","2000-01-01 01:20:18.408975","2000-01-01 01:31:35.457803","2000-01-01 01:42:52.506631","2000
-01-01 01:54:09.555459","2000-01-01 02:05:26.604287","2000-01-01 02:16:43.653115","2000-01-01 02:28:00.70
1943","2000-01-01 02:39:17.750771","2000-01-01 02:50:34.799599","2000-01-01 03:01:51.848427","2000-01-01
03:13:08.897255","2000-01-01 03:24:25.946085"]}}'::trajectory as a
)
select unnest(ST_split(a, '{"cut_point.max_point":4}')) from traj;

unnest

{"trajectory":{"version":1,"type":"STPOINT","leafcount":5,"start_time":"2000-01-01 00:01:19","end_time":
"2000-01-01 00:46:27","spatial":"LINESTRING(-100 -100 -100,-88.8925775739675 -86.6512698383691 -92.3767
832526937,-79.6904716538265 -80.6515727923252 -84.2357598245144,-75.8435507711644 -73.757289092832
6 -80.5007370118983,-70.6238425321256 -67.8213750167439 -74.5733173238113)","timeline":["2000-01-01 00
:01:19","2000-01-01 00:12:36","2000-01-01 00:23:53","2000-01-01 00:35:10","2000-01-01 00:46:27"]}}
{"trajectory":{"version":1,"type":"STPOINT","leafcount":5,"start_time":"2000-01-01 00:46:27","end_time":
"2000-01-01 01:31:35","spatial":"LINESTRING(-70.6238425321256 -67.8213750167439 -74.5733173238113,-61.
6014582272619 -61.0636760429479 -67.9874239303172,-56.1098577060426 -54.4264591250879 -64.500797204
6733,-46.9800617334743 -49.4026757289345 -61.6160059720278,-41.7122942996211 -46.3224360072054 -56.5
283147455193)","timeline":["2000-01-01 00:46:27","2000-01-01 00:57:44","2000-01-01 01:09:01","2000-01-01
01:20:18","2000-01-01 01:31:35"]}}
{"trajectory":{"version":1,"type":"STPOINT","leafcount":5,"start_time":"2000-01-01 01:31:35","end_time":
"2000-01-01 02:16:44","spatial":"LINESTRING(-41.7122942996211 -46.3224360072054 -56.5283147455193,-35.
5646221285375 -38.1688933617746 -49.2775720101781,-31.7230528349367 -33.6970051738123 -44.169371088
5011,-23.1585765127093 -26.5895827477798 -40.6539742602035,-16.7020264320696 -21.6133877349397 -37.3
055470525287)","timeline":["2000-01-01 01:31:35","2000-01-01 01:42:53","2000-01-01 01:54:10","2000-01-01
02:05:27","2000-01-01 02:16:44"]}}
{"trajectory":{"version":1,"type":"STPOINT","leafcount":5,"start_time":"2000-01-01 02:16:44","end_time":
"2000-01-01 03:01:52","spatial":"LINESTRING(-16.7020264320696 -21.6133877349397 -37.3055470525287,-12.
1044529232507 -14.1236051704424 -28.2295028120279,-3.77185660181567 -7.74744770256802 -24.384211162
1052,0.488159407706304 -3.68223926316326 -19.9478872027248,6.33406881305078 4.54123636645575 -15.04
10129944794)","timeline":["2000-01-01 02:16:44","2000-01-01 02:28:01","2000-01-01 02:39:18","2000-01-01 0
2:50:35","2000-01-01 03:01:52"]}}
{"trajectory":{"version":1,"type":"STPOINT","leafcount":3,"start_time":"2000-01-01 03:01:52","end_time":
"2000-01-01 03:24:26","spatial":"LINESTRING(6.33406881305078 4.54123636645575 -15.0410129944794,15.66
66049417108 10.5611746329814 -11.2770220567472,14 11 -10)","timeline":["2000-01-01 03:01:52","2000-01-0
1 03:13:09","2000-01-01 03:24:26"]}}
(5 rows)

7.4. Attribute metadata

7.4.1. ST_attrDefinition

This function returns the definitions of attributes in a trajectory object.

Syntax

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

Parameters

Parameter	Description
traj	The trajectory object.

Examples

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

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

7.4.2. ST_attrSize

This function returns the number of attributes in a trajectory object.

Syntax

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

Parameters

Parameter	Description
traj	The trajectory object.

Examples

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

7.4.3. ST_attrName

This function returns the name of a trajectory attribute field or the names of all trajectory attribute fields.

Syntax

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

Parameters

Parameter	Description
traj	The trajectory object.
index	The attribute index. The value starts from 0.

Examples

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

7.4.4. ST_attrType

This function returns the data type of a trajectory attribute field.

Syntax

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

Parameters

Parameter	Description
traj	The trajectory object.
index	The attribute index.
name	The name of the attribute field.

Description

This function returns one of the following strings to indicate the data type of the trajectory attribute field:

- integer
- float
- string
- timestamp
- bool

Examples

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

7.4.5. ST_attrLength

This function returns the defined length of a trajectory attribute field.

Syntax

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

Parameters

Parameter	Description
traj	The trajectory object.

Parameter	Description
index	The attribute index.
name	The name of the attribute field.

Examples

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

7.4.6. ST_attrNullable

This function returns true if a trajectory attribute field can be null.

Syntax

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

Parameters

Parameter	Description
traj	The trajectory object.

Parameter	Description
index	The attribute index.
name	The name of the attribute field.

Examples

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

7.5. Event functions

7.5.1. ST_addEvent

This function adds an event to a trajectory object.

Syntax

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

Parameters

Parameter	Description
traj	The trajectory object.
event_type	The event type ID.
event_time	The event timestamp.

Description

Event type IDs are predefined. For example, 1000 indicates unlocking and 2000 indicates locking.

Examples

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

st_addevent

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

7.5.2. ST_eventTimes

This function returns the time of all events in a trajectory object.

Syntax

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

Parameters

Parameter	Description
traj	The trajectory object.

Examples

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

7.5.3. ST_eventTime

This function returns the time of an event with the specified index in a trajectory object.

Syntax

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

Parameters

Parameter	Description
traj	The trajectory object.
index	The event index.

Examples

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

7.5.4. ST_eventTypes

This function returns the type IDs of all events in a trajectory object.

Syntax

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

Parameters

Parameter	Description
traj	The trajectory object.

Examples

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

7.5.5. ST_eventType

This function returns the type ID of an event with the specified index in a trajectory object.

Syntax

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

Parameters

Parameter	Description
traj	The trajectory object.
index	The event index.

Examples

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

7.6. Attribute functions

7.6.1. ST_startTime

This function returns the start time of a trajectory object.

Syntax

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

Parameters

Parameter	Description
traj	The trajectory object.

Examples

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

7.6.2. ST_endTime

This function returns the end time of a trajectory object.

Syntax

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

Parameters

Parameter	Description
traj	The trajectory object.

Examples

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

7.6.3. ST_trajectorySpatial

This function returns the spatial geometry object of a trajectory object.

Syntax

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

Parameters

Parameter	Description
traj	The trajectory object.

Example

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

7.6.4. ST_trajectoryTemporal

This function returns the timeline of a trajectory object.

Syntax

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

Parameters

Parameter	Description
traj	The trajectory object.

Examples

```
select ST_trajectoryTemporal(ST_MakeTrajectory('STPOINT'::leaftype, st_geomfromtext('LINESTRING (114
35, 115 36, 116 37)', 4326), '[2010-01-01 14:30, 2010-01-01 15:30)'::tsrange, null));
                    st_trajectorytemporal
-------------------------------------------------------------------------------
 {"timeline":["2010-01-01 14:30:00","2010-01-01 15:00:00","2010-01-01 15:30:00"]}
(1 row)
```

7.6.5. ST_trajAttrs

This function returns the attributes of a trajectory object.

Syntax

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

Parameters

Parameter	Description
traj	The trajectory object.

Examples

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

7.6.6. ST_attrIntMax

This function returns the maximum value of an integer-type attribute field.

Syntax

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

Parameters

Parameter	Description
traj	The trajectory object.

Parameter	Description
attr_field_name	The name of the attribute field.

Examples

```
select st_attrIntMax(ST_makeTrajectory('STPOINT'::leaftype, 'LINESTRING(-179.48077 51.72814,-179.46731 51.74634,-179.46502 51.74934,-179.46183 51.75378,-179.45943 51.75736,-179.45560 51.76273,-179.44845 51.77186,-179.43419 51.78977,-179.41259 51.81643,-179.41001 51.81941,-179.40751 51.82223,-179.40497 51.82505,-179.40242 51.82796,-179.39981 51.83095,-179.39734 51.83398,-179.39499 51.83709)'::geometry, ARRAY['2017-01-15 09:06:39'::timestamp,'2017-01-15 09:14:48','2017-01-15 09:13:39','2017-01-15 09:16:28','2017-01-15 09:19:48','2017-01-15 09:17:48','2017-01-15 09:23:19','2017-01-15 09:34:40','2017-01-15 09:30:28','2017-01-15 09:36:59','2017-01-15 09:38:09','2017-01-15 09:39:18','2017-01-15 09:40:40','2017-01-15 09:47:38','2017-01-15 21:18:30','2017-01-15 09:48:49'], '{"leafcount": 16, "attributes": {"heading": {"type": "integer", "length": 4, "nullable": false,"value":[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]}}}'), 'heading');
 st_attrintmax
---------------
        15
```

7.6.7. ST_attrIntMin

This function returns the minimum value of an integer-type attribute field.

Syntax

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

Parameters

Parameter	Description
traj	The trajectory object.
attr_field_name	The name of the attribute field.

Examples

```
select st_attrIntMin(ST_makeTrajectory('STPOINT'::leaftype, 'LINESTRING(-179.48077 51.72814,-179.46731 51.74634,-179.46502 51.74934,-179.46183 51.75378,-179.45943 51.75736,-179.45560 51.76273,-179.44845 51.77186,-179.43419 51.78977,-179.41259 51.81643,-179.41001 51.81941,-179.40751 51.82223,-179.40497 51.82505,-179.40242 51.82796,-179.39981 51.83095,-179.39734 51.83398,-179.39499 51.83709)'::geometry, ARRAY['2017-01-15 09:06:39'::timestamp,'2017-01-15 09:14:48','2017-01-15 09:13:39','2017-01-15 09:16:28','2017-01-15 09:19:48','2017-01-15 09:17:48','2017-01-15 09:23:19','2017-01-15 09:34:40','2017-01-15 09:30:28','2017-01-15 09:36:59','2017-01-15 09:38:09','2017-01-15 09:39:18','2017-01-15 09:40:40','2017-01-15 09:47:38','2017-01-15 21:18:30','2017-01-15 09:48:49'], '{"leafcount": 16, "attributes": {"heading": {"type": "integer", "length": 4, "nullable": false,"value":[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]}}}'), 'heading');
 st_attrintmin
---------------
        0
```

7.6.8. ST_attrIntAverage

This function returns the average value of an integer-type attribute field.

Syntax

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

Parameters

Parameter	Description
traj	The trajectory object.
attr_field_name	The name of the attribute field.

Examples

```
select st_attrIntAverage(ST_makeTrajectory('STPOINT'::leaftype, 'LINESTRING(-179.48077 51.72814,-179.46731 51.74634,-179.46502 51.74934,-179.46183 51.75378,-179.45943 51.75736,-179.45560 51.76273,-179.44845 51.77186,-179.43419 51.78977,-179.41259 51.81643,-179.41001 51.81941,-179.40751 51.82223,-179.40497 51.82505,-179.40242 51.82796,-179.39981 51.83095,-179.39734 51.83398,-179.39499 51.83709)'::geometry, ARRAY['2017-01-15 09:06:39'::timestamp,'2017-01-15 09:14:48','2017-01-15 09:13:39','2017-01-15 09:16:28','2017-01-15 09:19:48','2017-01-15 09:17:48','2017-01-15 09:23:19','2017-01-15 09:34:40','2017-01-15 09:30:28','2017-01-15 09:36:59','2017-01-15 09:38:09','2017-01-15 09:39:18','2017-01-15 09:40:40','2017-01-15 09:47:38','2017-01-15 21:18:30','2017-01-15 09:48:49'], '{"leafcount": 16, "attributes": {"heading": {"type": "integer", "length": 4, "nullable": false,"value":[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]}}}'), 'heading');
 st_attrintaverage
--------------------
                  7
```

7.6.9. ST_attrFloatMax

This function returns the maximum value of a float-type attribute field.

Syntax

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

Parameters

Parameter	Description
traj	The trajectory object.
attr_field_name	The name of the attribute field.

Examples

```
select st_attrFloatMax(ST_makeTrajectory('STPOINT'::leaftype, 'LINESTRING(-179.48077 51.72814,-179.46731 51.74634,-179.46502 51.74934,-179.46183 51.75378,-179.45943 51.75736,-179.45560 51.76273,-179.44845 51.77186,-179.43419 51.78977,-179.41259 51.81643,-179.41001 51.81941,-179.40751 51.82223,-179.40497 51.82505,-179.40242 51.82796,-179.39981 51.83095,-179.39734 51.83398,-179.39499 51.83709)'::geometry, ARRAY['2017-01-15 09:06:39'::timestamp,'2017-01-15 09:13:39','2017-01-15 09:14:48','2017-01-15 09:16:28','2017-01-15 09:17:48','2017-01-15 09:19:48','2017-01-15 09:23:19','2017-01-15 09:30:28','2017-01-15 09:34:40','2017-01-15 09:36:59','2017-01-15 09:38:09','2017-01-15 09:39:18','2017-01-15 09:40:40','2017-01-15 09:47:38','2017-01-15 09:48:49','2017-01-15 21:18:30'], '{"leafcount": 16, "attributes": {"heading": {"type": "float", "length": 4, "nullable": false,"value": [23.0,23.0,23.0,23.0,21.0,21.0,72.0,72.0,72.0,72.0,73.0,74.0,73.0,73.0,73.0,73.0]}}}'),'heading');
 st_attrfloatmax
-----------------
              74
```

7.6.10. ST_attrFloatMin

This function returns the minimum value of a float-type attribute field.

Syntax

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

Parameters

Parameter	Description
traj	The trajectory object.
attr_field_name	The name of the attribute field.

Examples

```
select st_attrFloatMin(ST_makeTrajectory('STPOINT'::leaftype, 'LINESTRING(-179.48077 51.72814,-179.46731 51.74634,-179.46502 51.74934,-179.46183 51.75378,-179.45943 51.75736,-179.45560 51.76273,-179.44845 51.77186,-179.43419 51.78977,-179.41259 51.81643,-179.41001 51.81941,-179.40751 51.82223,-179.40497 51.82505,-179.40242 51.82796,-179.39981 51.83095,-179.39734 51.83398,-179.39499 51.83709)'::geometry, ARRAY['2017-01-15 09:06:39'::timestamp,'2017-01-15 09:13:39','2017-01-15 09:14:48','2017-01-15 09:16:28','2017-01-15 09:17:48','2017-01-15 09:19:48','2017-01-15 09:23:19','2017-01-15 09:30:28','2017-01-15 09:34:40','2017-01-15 09:36:59','2017-01-15 09:38:09','2017-01-15 09:39:18','2017-01-15 09:40:40','2017-01-15 09:47:38','2017-01-15 09:48:49','2017-01-15 21:18:30'], '{"leafcount": 16, "attributes": {"heading": {"type": "float", "length": 4, "nullable": false,"value": [23.0,23.0,23.0,23.0,21.0,21.0,72.0,72.0,72.0,72.0,73.0,74.0,73.0,73.0,73.0,73.0]}}}'),'heading');
 st_attrfloatmin
-----------------
              21
```

7.6.11. ST_attrFloatAverage

This function returns the average value of a float-type attribute field.

Syntax

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

Parameters

Parameter	Description
traj	The trajectory object.
attr_field_name	The name of the attribute field.

Examples

```
select st_attrFloatAverage(ST_makeTrajectory('STPOINT'::leaftype, 'LINESTRING(-179.48077 51.72814,-179.46731 51.74634,-179.46502 51.74934,-179.46183 51.75378,-179.45943 51.75736,-179.45560 51.76273,-179.44845 51.77186,-179.43419 51.78977,-179.41259 51.81643,-179.41001 51.81941,-179.40751 51.82223,-179.40497 51.82505,-179.40242 51.82796,-179.39981 51.83095,-179.39734 51.83398,-179.39499 51.83709)'::geometry, ARRAY['2017-01-15 09:06:39'::timestamp,'2017-01-15 09:13:39','2017-01-15 09:14:48','2017-01-15 09:16:28','2017-01-15 09:17:48','2017-01-15 09:19:48','2017-01-15 09:23:19','2017-01-15 09:30:28','2017-01-15 09:34:40','2017-01-15 09:36:59','2017-01-15 09:38:09','2017-01-15 09:39:18','2017-01-15 09:40:40','2017-01-15 09:47:38','2017-01-15 09:48:49','2017-01-15 21:18:30'], '{"leafcount": 16, "attributes": {"heading": {"type": "float", "length": 4, "nullable": false,"value": [23.0,23.0,23.0,23.0,21.0,21.0,72.0,72.0,72.0,72.0,73.0,74.0,73.0,73.0,73.0,73.0]}}}'),'heading');
 st_attrfloataverage
---------------------
             53.8125
(1 row)
```

7.6.12. ST_leafType

This function returns the leaf type of a trajectory object.

Syntax

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

Parameters

Parameter	Description
traj	The trajectory object.

Only the ST_POINT type is supported.

Examples

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

7.6.13. ST_leafCount

This function returns the number of leaves (trajectory points) in a trajectory object.

Syntax

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

Parameters

Parameter	Description
traj	The trajectory object.

Examples

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

7.6.14. ST_duration

This function returns the duration of a trajectory object.

Syntax

```
interval ST_duration(trajectory traj);
```

Parameters

Parameter	Description
traj	The trajectory object.

Examples

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

7.6.15. ST_timeAtPoint

This function returns a set of time points when a trajectory object passes through the specified spatial point.

Syntax

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

Parameters

Parameter	Description
traj	The trajectory object.
g	The spatial point.

Example

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

7.6.16. ST_pointAtTime

This function returns the spatial geometry object of a trajectory object at the specified time point.

Syntax

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

Parameters

Parameter	Description
traj	The trajectory object.
t	The time point.

This function returns a spatial point.

Examples

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

7.6.17. ST_velocityAtTime

This function returns the value of the velocity attribute field at the specified time point.

Syntax

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

Parameters

Parameter	Description
traj	The trajectory object.
t	The time point.

Examples

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

7.6.18. ST_accelerationAtTime

This function returns the value of the acceleration attribute field at the specified time point.

Syntax

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

Parameters

Parameter	Description
traj	The trajectory object.
t	The specified time point.

Example

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

7.6.19. ST_timeToDistance

This function returns a line chart in which the time is the x-axis and the Euclidean distance is the y-axis.

Syntax

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

Parameters

Parameter	Description
traj	The trajectory object.

Examples

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

7.6.20. ST_timeAtDistance

This function returns the time point when the movement arrives at a spatial point from the start point for the specified distance.

Syntax

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

Parameters

Parameter	Description
traj	The trajectory object.
d	The distance of movement.

This function returns an array of timestamps. Multiple spatial points may be at the same distance from the start point.

Examples

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

7.6.21. ST_cumulativeDistanceAtTime

This function returns the cumulative distance of movement from the start point to the spatial point arrived at the specified time point.

Syntax

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

Parameters

Parameter	Description
traj	The trajectory object.
t	The time point.

Examples

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

7.6.22. ST_timeAtCumulativeDistance

This function returns the time point when the movement arrives at a spatial point from the start point for the specified cumulative distance.

Syntax

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

Parameters

Parameter	Description
traj	The trajectory object.
d	The cumulative distance of movement.

Examples

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

7.6.23. ST_subTrajectory

This function returns the sub-trajectory of a trajectory object within the specified time range.

Syntax

```
trajectory ST_subTrajectory(trajectory traj, timestamp starttime, timestamp endtime);□
trajectory ST_subTrajectory(trajectory traj, tsrange range);
```

Parameters

Parameter	Description
traj	The trajectory object.

Parameter	Description
starttime	The start time.
endtime	The end time.
range	The time range.

Examples

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

7.6.24. ST_subTrajectorySpatial

This function returns the spatial geometry object of a trajectory object within the specified time range.

Syntax

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

Parameters

Parameter	Description
traj	The trajectory object.
starttime	The start time.
endtime	The end time.
range	The time range.

Examples

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

7.6.25. ST_samplingInterval

This function returns the sampling interval.

Syntax

```
interval ST_samplingInterval(trajectory traj);
```

Parameters

Parameter	Description
traj	The trajectory object.

Examples

```
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)
```

7.6.26. ST_trajAttrsAsText

This function returns an array of values for a text-type attribute field of a trajectory object.

Syntax

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

Parameters

Parameter	Description
traj	The trajectory object.
attr_name	The name of the attribute field.

Examples

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

7.6.27. ST_trajAttrsAsInteger

This function returns an array of values for an integer-type attribute field of a trajectory object.

Syntax

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

Parameters

Parameter	Description
traj	The trajectory object.
attr_name	The name of the attribute field.

Examples

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

7.6.28. ST_trajAttrsAsDouble

This function returns an array of values for a double-type attribute field of a trajectory object.

Syntax

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

Parameters

Parameter	Description
traj	The trajectory object.
attr_name	The name of the attribute field.

Examples

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

7.6.29. ST_trajAttrsAsBool

This function returns an array of values for a Boolean-type attribute field of a trajectory object.

Syntax

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

Parameters

Parameter	Description
traj	The trajectory object.
attr_name	The name of the attribute field.

Examples

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

7.6.30. ST_trajAttrsAsTimestamp

This function returns an array of values for a timestamp-type attribute field of a trajectory object.

Syntax

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

Parameters

Parameter	Description
traj	The trajectory object.
attr_name	The name of the attribute field.

Examples

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

7.6.31. ST_attrIntFilter

You can specify the trajectory attribute field. This function filters out the trajectory points that meet the condition based on a fixed value, and returns the filtered attribute field values (array).

Syntax

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

Parameters

Parameter	Description
traj	The trajectory object attr
attr_field_name	The specified attribute name
operator	The filter operator: '=' , '! =' , '>' , '<' , '>=' , '<=' , '[]' , '()' , '[)' , or '()'
value, value1	The minimum fixed value of the attribute

Parameter	Description
value2	The maximum fixed value of the attribute

Description

Only the attribute fields of the integer type are supported.

Examples

```
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,-179.45560 51.76273,-179.44845 51.77186,-179.43419 51.78977,-179.41259 51.81643,-179.41001 51.81941,-179.40751 51.82223,-179.40497 51.82505,-179.40242 51.82796,-179.39981 51.83095,-179.39734 51.83398,-179.39499 51.83709)'::geometry, ARRAY['2017-01-15 09:06:39'::timestamp,'2017-01-15 09:14:48','2017-01-15 09:13:39','2017-01-15 09:16:28','2017-01-15 09:19:48','2017-01-15 09:17:48','2017-01-15 09:23:19','2017-01-15 09:34:40','2017-01-15 09:30:28','2017-01-15 09:36:59','2017-01-15 09:38:09','2017-01-15 09:39:18','2017-01-15 09:40:40','2017-01-15 09:47:38','2017-01-15 21:18:30','2017-01-15 09:48:49'], '{"leafcount": 16, "attributes" : {"heading" : {"type": "integer", "length": 4, "nullable" : false,"value":[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]}}}'));
select st_attrIntFilter(traj, 'heading', '>', 5) from traj where id = 1;
   st_attrintfilter
----------------------------
 {6,7,8,9,10,11,12,13,14,15}
(1 row)
select st_attrIntFilter(traj, 'heading', '>=', 5) from traj where id = 1;
    st_attrintfilter
-----------------------------
 {5,6,7,8,9,10,11,12,13,14,15}
(1 row)
select st_attrIntFilter(traj, 'heading', '<', 5) from traj where id = 1;
 st_attrintfilter
------------------
 {0,1,2,3,4}
(1 row)
select st_attrIntFilter(traj, 'heading', '<=', 5) from traj where id = 1;
 st_attrintfilter
------------------
 {0,1,2,3,4,5}
(1 row)
select st_attrIntFilter(traj, 'heading', '=', 5) from traj where id = 1;
 st_attrintfilter
------------------
 {5}
(1 row)
select st_attrIntFilter(traj, 'heading', '! =', 5) from traj where id = 1;
       st_attrintfilter
--------------------------------------
 {0,1,2,3,4,6,7,8,9,10,11,12,13,14,15}
(1 row)
select st_attrIntFilter(traj, 'heading', '()', 5,8) from traj where id = 1;
 st_attrintfilter
------------------
 {6,7}
```

```
{0,1}
(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)
```

7.6.32. ST_attrFloatFilter

You can specify the trajectory attribute field. This function filters out the trajectory points that meet the condition based on a fixed value, and returns the filtered attribute field values (array).

Syntax

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

Parameters

Parameter	Description
traj	The trajectory object
attr_field_name	The specified attribute name
operator	The filter operator: `'='` , `'! ='` , `'>'` , `'<'` , `'>='` , `'<='` , `'[]'` , `'(]'` , `'[)'` , and `'()'`
value, value1	The minimum fixed value of the attribute
value2	The maximum fixed value of the attribute

Description

Only the attribute fields of the float type are supported.

Examples

```
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)
```

7.6.33. ST_attrTimestampFilter

This function filters all trajectory points to obtain the trajectory points whose values for the specified attribute field meet a filter condition based on a fixed value or a value range and returns an array of the attribute field values of these points.

Syntax

```
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);
```

Parameters

Parameter	Description
traj	The trajectory object.
attr_field_name	The name of the specified attribute field.
operator	The filter operator. Valid values: '=' , '!=' , '>' , '<' , '>=' , '<=' , '[]' , '(]' , '[)' , '()' .
value and value1	The fixed value of the attribute field and the minimum fixed value of the attribute field.
value2	The maximum fixed value of the attribute field.

Examples

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

7.6.34. ST_attrNullFilter

This function filters all trajectory points to obtain the trajectory points whose values are null for the specified attribute field and returns a new trajectory object that is composed of these points.

Syntax

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

Parameters

Parameter	Description
traj	The trajectory object.
attr_field_name	The name of the specified attribute field.

Description

This function supports all types of attribute fields.

Examples

```
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)
```

7.6.35. ST_attrNotNullFilter

This function filters all trajectory points to obtain the trajectory points whose values are not null for the specified attribute field and returns a new trajectory object that is composed of these points.

Syntax

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

Parameters

Parameter	Description
traj	The trajectory object.
attr_field_name	The name of the specified attribute field.

Description

This function supports all types of attribute fields.

Examples

```
select st_attrNotNullFilter(ST_makeTrajectory('STPOINT'::leaftype, 'LINESTRING(-179.48077 51.72814,-179.46731 51.74634,-179.46502 51.74934,-179.46183 51.75378,-179.45943 51.75736,-179.45560 51.76273,-179.44845 51.77186,-179.43419 51.78977,-179.41259 51.81643,-179.41001 51.81941,-179.40751 51.82223,-179.40497 51.82505,-179.40242 51.82796,-179.39981 51.83095,-179.39734 51.83398,-179.39499 51.83709)'::geometry, ARRAY['2017-01-15 09:06:39'::timestamp,'2017-01-15 09:13:39','2017-01-15 09:14:48','2017-01-15 09:16:28','2017-01-15 09:17:48','2017-01-15 09:19:48','2017-01-15 09:23:19','2017-01-15 09:30:28','2017-01-15 09:34:40','2017-01-15 09:36:59','2017-01-15 09:38:09','2017-01-15 09:39:18','2017-01-15 09:40:40','2017-01-15 09:47:38','2017-01-15 09:48:49','2017-01-15 21:18:30'], '{"leafcount": 16, "attributes": {"heading": {"type": "float", "length": 4, "nullable": true,"value": [23.0,23.0,23.0,null,21.0,21.0,null,72.0,72.0,null,73.0,74.0,73.0,73.0,null,73.0]}}}'),'heading');
                st_attrnotnullfilter

-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
 {"trajectory":{"version":1,"type":"STPOINT","leafcount":12,"start_time":"2017-01-15 09:06:39","end_time":"2017-01-15 21:18:30","spatial":"LINESTRING(-179.48077 51.72814,-179.46731 51.74634,-179.46502 51.74934,-179.45943 51.75736,-179.4556 51.76273,-179.43419 51.78977,-179.41259 51.81643,-179.40751 51.82223,-179.40497 51.82505,-179.40242 51.82796,-179.39981 51.83095,-179.39499 51.83709)","timeline":["2017-01-15 09:06:39","2017-01-15 09:13:39","2017-01-15 09:14:48","2017-01-15 09:17:48","2017-01-15 09:19:48","2017-01-15 09:30:28","2017-01-15 09:34:40","2017-01-15 09:38:09","2017-01-15 09:39:18","2017-01-15 09:40:40","2017-01-15 09:47:38","2017-01-15 21:18:30"],"attributes":{"leafcount":12,"heading":{"type":"float","length":4,"nullable":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)
```

7.6.36. ST_trajAttrsMeanMax

This function returns the maximum average value of an attribute field by using the MEAN-MAX algorithm.

Syntax

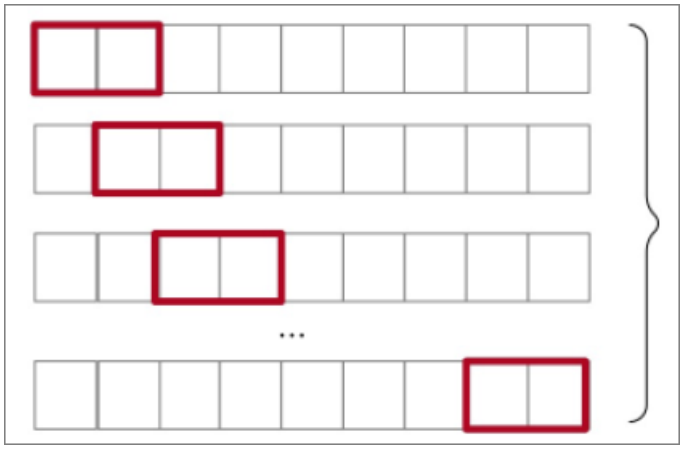
```
SETOF record ST_trajAttrsMeanMax(trajectory traj, cstring attr_field_name, out interval duration, out float8 max);
```

Parameters

Parameter	Description
traj	The trajectory object.
attr_field_name	The name of the specified attribute field.

Description

The MEAN-MAX algorithm uses a sliding window to compute the average value of the specified attribute field in each time range specified by the window, and then returns the maximum value of all the average values.



This function supports only attribute fields of the integer and float types. The values of the specified attribute field cannot be null.

Examples

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

7.7. Functions to process bounding boxes

7.7.1. ST_MakeBox

This function is used to query the bounding box of an object over a specified time range.

Syntax

```
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);
```

Parameters

Parameter	Description
geom	The geometry whose bounding box you want to query.
traj	The trajectory whose bounding box you want to query.
ts	The beginning of the time range to query.
te	The end of the time range to query.

Description

A bounding box is the minimal multi-dimensional rectangle that contains a spatio-temporal object.

- If you specify a two- or three-dimensional spatial object, this function returns the bounding box of the object based on the dimensions of the object.
- If you specify a spatial object and a time range by using the `ST_MakeBox(geometry geom, timestamp ts, timestamp te)` syntax, this function returns the bounding box of the object over the time range.
- If you specify a trajectory but do not specify a time range, this function returns the bounding box of the complete trajectory. If you specify a trajectory and a time range, this function returns the bounding box of the sub-trajectory over the time range.
- Bounding boxes use the FLOAT data type. Therefore, this function may return a bounding box that is slightly larger than specified by the parameter settings. For example, the returned lower bound is slightly smaller than the actual lower bound, or the returned upper bound is slightly larger than the actual upper bound.

Example

```
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)
```

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

This function is used to build a bounding box based on specified parameter settings.

Syntax

```
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, timestamp 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, float8 ymax, float8 zmax, timestamp tmax);
```

Parameters

Parameter	Description
xmin	The lower bound on the x axis of the bounding box.
xmax	The upper bound on the x axis of the bounding box.

Parameter	Description
ymin	The lower bound on the y axis of the bounding box.
ymax	The upper bound on the y axis of the bounding box.
zmin	The lower bound on the z axis of the bounding box.
zmax	The upper bound on the z axis of the bounding box.
tmin	The lower bound on the t axis of the bounding box.
tmax	The upper bound on the t axis of the bounding box.

Description

This function allows you to build a bounding box based on the function name and the specified parameter settings.

Bounding boxes use the FLOAT data type. Therefore, this function may return a bounding box that is slightly larger than specified by the parameter settings. For example, the returned lower bound is slightly smaller than the actual lower bound, or the returned upper bound is slightly larger than the actual upper bound.

Example

```
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)
```

7.7.3. ST_BoxndfToGeom

This function is used to convert a bounding box into a geometry.

Syntax

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

Parameters

Parameter	Description
box	The bounding box that you want to convert into a geometry.

Description

This function allows you to convert a bounding box into a geometry.

- If the bounding box has the z dimension, this function converts the bounding box into a three-dimensional geometry.
- If the bounding box does not have the z dimension, this function converts the bounding box into a two-dimensional geometry.
- If the bounding box does not have the x or y dimension, this function returns NULL.

Example

```
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))
```

7.7.4. ST_Has{xy|z|t}

The ST_Has{xy|z|t} function is used to check whether a bounding box has one or more specified dimensions.

Syntax

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

Parameters

Parameter	Description
box	The bounding box that you want to query.

Description

This function allows you to check whether a bounding box has one or more specified dimensions. The x and y dimensions are bound to each other. Therefore, the bounding box has both or neither of the x and y dimensions.

Example

```
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
```

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

The function is used to query the minimum value of a bounding box in a specified dimension.

Syntax

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

Parameters

Parameter	Description
box	The bounding box whose minimum value in a specified dimension you want to query.

Description

This function allows you to query the minimum value of a bounding box in a specified dimension.

- If the bounding box does not have the specified x, y, or z dimension, this function returns `NaN`.
- If the bounding box does not have the specified t dimension, this function returns `-infinity`.

Example

```
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
```

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

This function is used to query the maximum value of a bounding box in a specified dimension.

Syntax

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

Parameters

Parameter	Description
box	The bounding box whose maximum value in a specified dimension you want to query.

Description

This function allows you to query the maximum value of a bounding box in a specified dimension.

- If the bounding box does not have the specified x, y, or z dimension, this function returns `NaN`.
- If the bounding box does not have the specified t dimension, this function returns `-infinity`.

Example

```
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
```

7.7.7. ST_ExpandSpatial

This function is used to expand a bounding box to a specified spatial scope.

Syntax

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

Parameters

Parameter	Description
box	The bounding box that you want to expand.
len	The length to which you want to expand the bounding box.

Description

This function allows you to expand a bounding box to a specified spatial scope. If the bounding box has the x, y, and z dimensions, the lower bound is decreased by the value of the len parameter, and the upper bound is increased by the value of the len parameter.

Example

```
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)
```

7.8. Operators to process bounding boxes

7.8.1. Basic concepts

A bounding box-processing operator is used to check whether the bounding boxes of the two objects that are specified by the left and right operands meet a specified spatio-temporal relationship.

Operator marks

The following operator marks are supported:

- The mark for INTERSECT operators: `&&` or `&`
- The mark for INCLUDE operators: `@>`
- The mark for INCLUDED operators: `<@`

> **Note**
>
> - If a dimension mark resides between a pair of operator marks, the operator compares two objects in this dimension in addition to the x and y dimensions.
> - If a dimension mark resides on the left and right sides of an operator mark, the operator compares two objects only in this dimension. In this case, the operator does not compare the two objects in the other dimensions, such as the x and y dimensions.

Dimension marks

The following dimension marks are supported:

- The mark for the z dimension: /
- The mark for the t dimension: #

> **Note**
>
> - If an operator does not contain a dimension mark, the operator specifies a two-dimensional relationship.
> - If an operator contains both dimension marks, the mark for the z dimension precedes the mark for the t dimension.

7.8.2. INTERSECT operators

This topic describes the INTERSECT operators. These operators are used to check whether the bounding box of the left operand-specified object intersects with the bounding box of the right operand-specified object in a specified dimension.

Syntax

```
{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}
```

Parameters

Parameter	Description
Left operand	The object whose bounding box you want to compare.
Right operand	The object whose bounding box you want to compare.

Description

The INTERSECT operators allow you to check whether the bounding box of the left operand-specified object intersects with the bounding box of the right operand-specified object in a specified dimension. These bounding boxes are generated by the ST_MakeBox function. For more information, see ST_MakeBox.

The following INTERSECT operators are supported:

- `/&/` : checks whether the bounding boxes of two objects intersect in the z dimension.
- `#&#` : checks whether the bounding boxes of two objects intersect in the t dimension.
- `&&` : checks whether the bounding boxes of two objects intersect in the x and y dimensions.
- `&/&` : checks whether the bounding boxes of two objects intersect in the x, y, and z dimensions.
- `&#&` : checks whether the bounding boxes of two objects intersect in the x, y, and t dimensions.
- `&/#&` : checks whether the bounding boxes of two objects intersect in the x, y, z, t, and t dimensions.

Example

```
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
```

7.8.3. INCLUDE operators

This topic describes the INCLUDE operators. These operators are used to check whether the bounding box of the left operand-specified object includes the bounding box of the right operand-specified object in a specified dimension.

Syntax

```
{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}
```

Parameters

Parameter	Description
Left operand	The object whose bounding box you want to compare.
Right operand	The object whose bounding box you want to compare.

Description

The INCLUDE operators allow you to check whether the bounding box of the left operand-specified object includes the bounding box of the right operand-specified object in a specified dimension. These bounding boxes are generated by the ST_MakeBox function. For more information, see ST_MakeBox. The effect of the INCLUDE operators is opposite to that of the INCLUDED operators. For more information, see INCLUDED operators.

The following INCLUDE operators are supported:

- `/@>/` : checks whether the bounding box of the left operand-specified object includes the bounding box of the right operand-specified object in the z dimension.
- `#@>#` : checks whether the bounding box of the left operand-specified object includes the bounding box of the right operand-specified object in the t dimension.
- `@>` : checks whether the bounding box of the left operand-specified object includes the bounding box of the right operand-specified object in the x and y dimensions.
- `@&>` : checks whether the bounding box of the left operand-specified object includes the bounding box of the right operand-specified object in the x, y, and z dimensions.
- `<#@` : checks whether the bounding box of the left operand-specified object is included in the bounding box of the right operand-specified object in the x, y, and t dimensions.
- `@/#>` : checks whether the bounding box of the left operand-specified object is included in the bounding box of the right operand-specified object in the x, y, z, and t dimensions.

Example

```
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
```

7.8.4. INCLUDED operators

This topic describes the INCLUDED operators. These operators are used to check whether the bounding box of the left operand-specified object is included in the bounding box of the right operand-specified object in a specified dimension.

Syntax

```
{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}
```

Parameters

Parameter	Description
Left operand	The object whose bounding box you want to compare.
Right operand	The object whose bounding box you want to compare.

Description

The INCLUDED operators allow you to check whether the bounding box of the left operand-specified object is included in the bounding box of the right operand-specified object in a specified dimension. These bounding boxes are generated by the ST_MakeBox function. For more information, see ST_MakeBox. The effect of the INCLUDED operators is opposite to that of the INCLUDE operators. For more information, see INCLUDE operators.

The following INCLUDED operators are supported:

- `/<@/` : checks whether the bounding box of the left operand-specified object is included in the bounding box of the right operand-specified object in the z dimension.
- `#<@#` : checks whether the bounding box of the left operand-specified object is included in the bounding box of the right operand-specified object in the t dimension.
- `<@` : checks whether the bounding box of the left operand-specified object is included in the bounding box of the right operand-specified object in the x and y dimensions.

- `<&@` : checks whether the bounding box of the left operand-specified object is included in the bounding box of the right operand-specified object in the x, y, and z dimensions.
- `<#@` : checks whether the bounding box of the left operand-specified object is included in the bounding box of the right operand-specified object in the x, y, and t dimensions.
- `</#@` : checks whether the bounding box of the left operand-specified object is included in the bounding box of the right operand-specified object in the x, y, z, and t dimensions.

Example

```
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
```

7.9. Spatial relationship judgment

7.9.1. ST_intersects

This function returns true if a trajectory object and the specified geometry object intersect in space within the specified time range.

Syntax

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

Parameters

Parameter	Description
traj	The trajectory object.
t1	The start time.
t2	The end time.
range	The time range.
g	The specified geometry object.

Examples

```
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;
```

7.9.2. ST_equals

This function returns true if a trajectory object and the specified geometry object are spatially equal within the specified time range.

Syntax

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

Parameters

Parameter	Description
traj	The trajectory object.
t1	The start time.
t2	The end time.
range	The time range.
g	The specified geometry object.

Example

```
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;
```

7.9.3. ST_distanceWithin

This function returns true if the distance between a trajectory object and the specified geometry object within the specified time range is within the reference distance.

Syntax

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

Parameters

Parameter	Description
traj	The trajectory object.
t1	The start time.
t2	The end time.

Parameter	Description
range	The time range.
g	The specified geometry object.
d	The reference distance.

Example

```
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;
```

7.10. Spatial processing

7.10.1. ST_intersection

This function returns a new trajectory object that indicates the intersection of a trajectory object and the specified geometry object within the specified time range.

Syntax

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

Parameters

Parameter	Description
traj	The trajectory object.
t1	The start time.
t2	The end time.
range	The time range.
g	The specified geometry object.

Description

If the trajectory object and the geometry object intersect at multiple spatial points, this function returns multiple sub-trajectories.

Examples

```
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;
```

7.10.2. ST_difference

This function returns a new trajectory object that indicates the difference between a trajectory object and the specified geometry object within the specified time range.

Syntax

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

Parameters

Parameter	Description
traj	The trajectory object.
t1	The start time.
t2	The end time.
range	The time range.
g	The specified geometry object.

Examples

```
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;
```

7.11. Spatial statistics

7.11.1. ST_nearestApproachPoint

This function returns the spatial point in a trajectory object that is nearest to the specified geometry object within the specified time range.

Syntax

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

Parameters

Parameter	Description
traj	The trajectory object.
t1	The start time.

Parameter	Description
t2	The end time.
range	The time range.
g	The specified geometry object.

Example

```
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;
```

7.11.2. ST_nearestApproachDistance

This function returns the nearest distance between a trajectory object and the specified geometry object within the specified time range.

Syntax

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

Parameters

Parameter	Description
traj	The trajectory object.
t1	The start time.
t2	The end time.
range	The time range.
g	The specified geometry object.

Example

```
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;
```

7.12. Spatio-temporal relationship judgment

7.12.1. ST_intersects

This function returns true if trajectory objects 1 and 2 intersect in space within the specified time range.

Syntax

```
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);
```

Parameters

Parameter	Description
traj	The trajectory object.
t1	The start time.
t2	The end time.
range	The time range.

Examples

```
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');
```

7.12.2. ST_equals

This function returns true if trajectory objects 1 and 2 are spatially equal within the specified time range.

Syntax

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

Parameters

Parameter	Description
traj	The trajectory object.
t1	The start time.
t2	The end time.
range	The time range.

Examples

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

7.12.3. ST_distanceWithin

This function returns true if the distance between trajectory objects 1 and 2 within the specified time range is within the reference distance.

Syntax

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

Parameters

Parameter	Description
traj	The trajectory object.
t1	The start time.
t2	The end time.
range	The time range.
d	The reference distance.

Examples

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

7.12.4. ST_durationWithin

This function returns true if the difference between the time when trajectory objects 1 and 2 intersect at the same spatial point within the specified time range is within the reference interval.

Syntax

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

Parameters

Parameter	Description
traj	The trajectory object.

Parameter	Description
t1	The start time.
t2	The end time.
range	The time range.
i	The reference interval.

Description

If two trajectory objects intersect at the same spatial point multiple times, this function returns true so long as any time difference is within the reference interval.

Examples

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

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

This function is used to check whether two objects intersect on a specified axis.

Syntax

```
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, timestamp te);
```

Parameters

Parameter	Description
geom	The geometry that you want to compare.
traj	The trajectory that you want to compare, or the original trajectory that includes the sub-trajectory that you want to compare.
traj1	The trajectory that you want to compare, or the original trajectory that includes the sub-trajectory that you want to compare.
traj2	The trajectory that you want to compare, or the original trajectory that includes the sub-trajectory that you want to compare.

Parameter	Description
box	The bounding box that you want to compare.
ts	The beginning of the time range over which you want to extract sub-trajectories. This parameter is optional.
te	The end of the time range over which you want to extract sub-trajectories. This parameter is optional.

Description

This function allows you to check whether two objects intersect in a specified dimension. If a dimension does not have a specified value, this function considers this dimension to have any value.

If this function contains the ts and te parameters, it compares two objects to check whether the sub-trajectories of the two objects over the specified time range intersect. If this function does not contain the ts or te parameter, it compares two objects to check whether the complete trajectories of the two objects intersect. The two objects are specified by the traj or traj1 parameter and the traj2 parameter.

The `ST_Intersects(...)` function delivers the same effect as the `ST_2DIntersects(...)` function (if one of the specified objects is a geometry) or the `ST_3DTIntersects(...)` function (if both the specified objects are trajectories).

Example

```
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":"2000-01-01 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 69.630026333034 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"]}}')::trajecto
ry 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
```

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

This function is used to determine whether two objects intersect on a specified axis. This function also allows you to specify to use the index on the column that stores the first specified object.

Syntax

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

Parameters

Parameter	Description
traj1	The trajectory that you want to compare, or the original trajectory that includes the sub-trajectory that you want to compare.
traj2	The trajectory that you want to compare, or the original trajectory that includes the sub-trajectory that you want to compare.
ts	The beginning of the time range over which you want to extract sub-trajectories. This parameter is optional.
te	The end of the time range over which you want to extract sub-trajectories. This parameter is optional.

Description

If you call the ST_ndIntersects function to compare two sub-trajectories by using the `ST_{2D|2DT|3D|3DT}Intersects(trajectory traj1, trajectory traj2, timestamp ts, timestamp te)` syntax, the function cannot determine which index to use. This causes additional computing overheads.

We recommend that you use the `ST_ndIntersects_IndexLeft(...)` function. This function allows you to manually specify to use the index on the column that stores the first specified sub-trajectory. This reduces computing overheads.

Example

```
EXPLAIN SELECT count(*) FROM sqltr_test_trajs WHERE ST_2DIntersects_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)'::tsr
ange,
 '{"leafcount":3,"attributes":{"velocity": {"type": "integer", "length": 2,"nullable" : true,"value": [120,130,14
0]}, "accuracy": {"type": "float", "length": 4, "nullable" : false,"value": [120,130,140]}, "bearing": {"type": "flo
at", "length": 8, "nullable" : false,"value": [120,130,140]}, "acceleration": {"type": "string", "length": 20, "null
able" : 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 15:00:0
0 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 &#& tr
aj)
     Filter: _st_2dintersects(traj, '{"trajectory":{"version":1,"type":"STPOINT","leafcount":3,"start_time":"2
000-01-01 00:01:30","end_time":"2000-01-01 00:15:00","spatial":"LINESTRING(-70 -30,-72 -34,-73 -35)","timeli
ne":["2000-01-01 00:01:30","2000-01-01 00:08:15","2000-01-01 00:15:00"],"attributes":{"leafcount":3,"velocit
y":{"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":[12
0.0,130.0,140.0]},"acceleration":{"type":"string","length":20,"nullable":true,"value":["120","130","140"]},"a
ctive":{"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, '20
00-01-01 00:00:00'::timestamp without 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 width=0)
        Index Cond: (traj &#& 'BOX2DT(-73 -35 2000-01-01 00:01:29.999994,-70 -30 2000-01-01 00:15:00)'::boxnd
f)
```

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

This function is used to check whether the distance between two objects on a specified axis is less than or equal to a specified threshold.

Syntax

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

Parameters

Parameter	Description
geom	The geometry that you want to compare.

Parameter	Description
traj	The trajectory that you want to compare, or the original trajectory that includes the sub-trajectory that you want to compare.
traj1	The trajectory that you want to compare, or the original trajectory that includes the sub-trajectory that you want to compare.
traj2	The trajectory that you want to compare, or the original trajectory that includes the sub-trajectory that you want to compare.
box	The bounding box that you want to compare.
ts	The beginning of the time range over which you want to extract sub-trajectories. This parameter is optional.
te	The end of the time range over which you want to extract sub-trajectories. This parameter is optional.
dist	The threshold for the distance between the two objects.

Description

- For 2D and 3D operations, this function checks whether the distance between the two- or three-dimensional projections of two objects is less than or equal to the specified threshold.
- For 2DT and 3DT operations, this function checks whether the distance between two objects in a specified dimension at a specified point in time is less than or equal to the specified threshold.

If this function contains the ts and te parameters, it compares two objects to check whether the distance between the sub-trajectories of the two objects over the specified time range is less than or equal to a specified threshold. If this function does not contain the ts or te parameter, it compares two objects to check whether the distance between the complete trajectories of the two objects is less than or equal to a specified threshold. The two objects are specified by the traj or traj1 parameter and the traj2 parameter.

The `ST_DistanceWithin(..)` function delivers the same effect as the `ST_2DDWithin(...)` function (if one of the specified objects is a geometry) and the `ST_3DTDWithin(...)` function (if both the specified objects are trajectories).

Example

```
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", "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,
      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", "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"}]}') b
)
SELECT ST_2dDWithin(a,b, ST_PGEpochToTS(0),ST_PGEpochToTS(49), 19), ST_3dDWithin(a,b, ST_PGEpoch
ToTS(0),ST_PGEpochToTS(49), 19),
    ST_2dDWithin(a,b, ST_PGEpochToTS(0),ST_PGEpochToTS(50), 20), ST_3dDWithin(a,b, ST_PGEpochToT
S(0),ST_PGEpochToTS(50), 20),
    ST_2dDWithin(a,b, ST_PGEpochToTS(0),ST_PGEpochToTS(70), 21), ST_3dDWithin(a,b, ST_PGEpochToT
S(0),ST_PGEpochToTS(70), 21),
    ST_2dtDWithin(a,b, ST_PGEpochToTS(0),ST_PGEpochToTS(49), 19), ST_3dtDWithin(a,b, ST_PGEpochTo
TS(0),ST_PGEpochToTS(49), 19),
    ST_2dtDWithin(a,b, ST_PGEpochToTS(0),ST_PGEpochToTS(50), 20), ST_3dtDWithin(a,b, ST_PGEpochTo
TS(0),ST_PGEpochToTS(50), 20),
    ST_2dtDWithin(a,b, ST_PGEpochToTS(0),ST_PGEpochToTS(70), 21), ST_3dtDWithin(a,b, ST_PGEpochTo
TS(0),ST_PGEpochToTS(70), 21) from traj;
NOTICE:  One or both of the geometries is missing z-value. The unknown z-value will be regarded as "any val
ue"
NOTICE:  One or both of the geometries is missing z-value. The unknown z-value will be regarded as "any val
ue"
 st_2ddwithin | st_3ddwithin | st_2ddwithin | st_3ddwithin | st_2ddwithin | st_3ddwithin | st_2dtdwithin | st
_3dtdwithin | st_2dtdwithin | st_3dtdwithin | st_2dtdwithin | st_3dtdwithin
--------------+--------------+--------------+--------------+--------------+--------------+---------------+---
-------------+---------------+---------------
 f            | f            | t            | t            | t            | t            | f             | f
            | t             | t             | t             | t
```

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

This function is used to check whether the distance between two objects on a specified axis is less than or equal to a specified threshold. This function also allows you to specify to use the index on the column that stores the first specified object.

Syntax

```
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, fl
oat8 dist);
```

Parameters

Parameter	Description
traj1	The trajectory that you want to compare, or the original trajectory that includes the sub-trajectory that you want to compare.
traj2	The trajectory that you want to compare, or the original trajectory that includes the sub-trajectory that you want to compare.
ts	The beginning of the time range over which you want to extract sub-trajectories. This parameter is optional.
te	The end of the time range over which you want to extract sub-trajectories. This parameter is optional.
dist	The threshold for the distance between the two objects.

Description

If you call the `ST_ndDWithin(...)` function to compare two trajectories by using the `ST_{2D|2DT|3D|3DT}DWithin(trajectory traj1, trajectory traj2, float8 dist)` or `ST_{2D|2DT|3D|3DT}DWithin(trajectory traj1, trajectory traj2, timestamp ts, timestamp te, float8 dist)` syntax, the function cannot determine which index to use. This causes additional computing overheads.

We recommend that you use the ST_ndDwithin_IndexLeft function. This function allows you to manually specify to use the index on the column that stores the first specified object. This reduces computing overheads.

Example

```
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)
```

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

This function is used to check whether the first specified object includes the second specified object on a specified axis.

Syntax

```
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);
```

Parameters

Parameter	Description
geom	The geometry that you want to compare.
traj	The trajectory that you want to compare, or the original trajectory that includes the sub-trajectory that you want to compare.

Parameter	Description
box	The bounding box that you want to compare.
r	The time range to query.
ts	The beginning of the time range over which you want to extract sub-trajectories. This parameter is optional.
te	The end of the time range over which you want to extract sub-trajectories. This parameter is optional.

Description

This function allows you to check whether the first specified object includes the second specified object on a specified axis.

- If you specify a geometry, this function compares the two-dimensional projections of the complete trajectories or the sub-trajectories over a specified time range. Then, this function checks whether the geometry includes or is included in the other specified object.
- If the first specified object is a bounding box and the second specified object is a trajectory, this function compares the bounding box and the trajectory to check whether the trajectory or its sub-trajectory over a specified time range is included in the bounding box in all dimensions. If the bounding box, trajectory, or sub-trajectory does not have a specified dimension, this function considers this dimension to have any value. In this case, the axis in this dimension automatically meets the specified condition.

> **Note** This function is not supported for geometry types such as POLYHEDRALSURFACE.

Example

```
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", "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"}]}') 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_PGEpochT
oTS(0), ST_PGEpochToTS(49)),
    ST_2dtContains(a,b,ST_PGEpochToTS(0), ST_PGEpochToTS(49)), ST_3dtContains(a,b,ST_PGEpochToTS
(0), ST_PGEpochToTS(49)),
    ST_2dContains(a,b,ST_PGEpochToTS(0), ST_PGEpochToTS(50)), ST_3dContains(a,b,ST_PGEpochToTS(
0), ST_PGEpochToTS(50)),
    ST_2dtContains(a,b,ST_PGEpochToTS(0), ST_PGEpochToTS(50)), ST_3dtContains(a,b,ST_PGEpochToTS
(0), ST_PGEpochToTS(50)),
    ST_2dContains(a,b,ST_PGEpochToTS(0), ST_PGEpochToTS(70)), ST_3dContains(a,b,ST_PGEpochToTS(
0), ST_PGEpochToTS(70)),
    ST_2dtContains(a,b,ST_PGEpochToTS(0), ST_PGEpochToTS(70)), ST_3dtContains(a,b,ST_PGEpochToTS
(0), ST_PGEpochToTS(70)) from traj;
 st_2dcontains | st_3dcontains | st_2dtcontains | st_3dtcontains | st_2dcontains | st_3dcontains | st_2dtcont
ains | st_3dtcontains | st_2dcontains | st_3dcontains | st_2dtcontains | st_3dtcontains
---------------+---------------+----------------+----------------+---------------+---------------+----------------+---------
------+---------------+----------------+----------------
t       | t       | t       | t       | t       | t       | f       | f       | f       | f       | f       | f
```

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

This topic describes the ST_{T|2D|2DT|3D|3DT}Within function, which is used to determine whether the first specified object is included in the second specified object on a specified axis.

Syntax

```
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);
```

Parameters

Parameter	Description
geom	The geometry that you want to compare.

Parameter	Description
traj	The trajectory that you want to compare, or the original trajectory that includes the sub-trajectory that you want to compare.
box	The bounding box that you want to compare.
r	The time range that you want to compare.
ts	The beginning of the time range over which you want to extract sub-trajectories. This parameter is optional.
te	The end of the time range over which you want to extract sub-trajectories. This parameter is optional.

Description

This function allows you to determine whether the first specified object is included in the second specified object. This function works in the same way as the ST_ndContains function.

> **Note** This function is not supported for geometry types such as POLYHEDRALSURFACE.

Example

```
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 47.230026333034 0,64.9789719
42887 60.618813472986 0,77.621717839502 78.012496630661 0,80 78 0)'::geometry b
)
SELECT ST_2dWithin(b,a) from traj;
 st_2dwithin
-------------
 t
```

7.13. Spatio-temporal processing

7.13.1. ST_intersection

This function returns a new trajectory object that indicates the intersection of trajectory objects 1 and 2 within the specified time range.

Syntax

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

Parameters

Parameter	Description
traj1	Trajectory object 1.
traj2	Trajectory object 2.
t1	The start time.
t2	The end time.
range	The time range.

Examples

```
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');
```

7.14. Spatio-temporal statistics

7.14.1. ST_nearestApproachPoint

This function returns the spatial point in trajectory object 1 that is nearest to trajectory object 2 within the specified time range.

Syntax

```
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);
```

Parameters

Parameter	Description
traj1	The trajectory object 1.
traj2	The trajectory object 2.
t1	The start time.
t2	The end time.
range	The time range.

Example

```
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');
```

7.14.2. ST_nearestApproachDistance

This function returns the nearest distance between trajectory objects 1 and 2 within the specified time range.

Syntax

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

Parameters

Parameter	Description
traj1	Trajectory object 1.
traj2	Trajectory object 2.
t1	The start time.
t2	The end time.
range	The time range.

Examples

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

7.15. Distance measurement

7.15.1. ST_length

This function returns the total length of a trajectory object, in meters.

Syntax

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

Parameters

Parameter	Description
traj	The trajectory object.
srid	The spatial reference system identifier (SRID) of the trajectory object. Default value: 0 (4326).

Description

This function computes the spherical length in meters based on the SRID of the trajectory object. You can specify the SRID for a trajectory object by setting the srid parameter. If not specified, the SRID is 4326 by default.

Examples

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

7.15.2. ST_euclideanDistance

This function returns the nearest Euclidean distance between two trajectory objects at the same time point.

Syntax

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

Parameters

Parameter	Description
traj1	The trajectory object 1.
traj2	The trajectory object 2.

Description

The distance has been standardized.

Examples

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

7.15.3. ST_mdistance

This function returns an array of the Euclidean distance between two trajectory objects at the same time point.

Syntax

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

Parameters

Parameter	Description
traj1	The trajectory object 1.
traj2	The trajectory object 2.

Description

The distance has not been standardized.

Examples

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

7.16. Similarity analysis

7.16.1. ST_lcsSimilarity

This function computes the similarity between two trajectory objects based on the longest common sub-sequence (LCSS) algorithm.

Syntax

```
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');
```

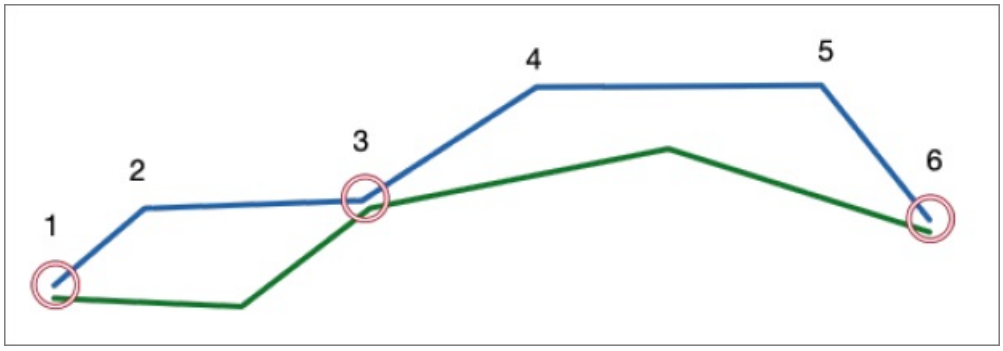
Parameters

Parameter	Description
traj1	The trajectory object 1.
traj2	The trajectory object 2.
dist	The distance tolerance between two trajectory points. Unit: meters.
lag	The time tolerance between two trajectory points.

Parameter	Description
unit	The unit of the distance. Valid values: • 'M': meters. • 'KM': kilometers. • 'D': degrees. This value is valid only when the spatial reference system identifier (SRID) of a trajectory object is WGS84 (4326 by default).

Description

The LCSS algorithm is used to compute the maximum similarity between two trajectory objects. The consistency between two trajectory points is determined based on their distance in space and time. This function computes the similarity between two trajectory objects and returns the number of consistent trajectory points.



As shown in the preceding figure, trajectory points 1, 3, and 6 meet the conditions of consistency. In this case, this function returns 3.

Generally, a trajectory object has a valid SRID. If not specified, the SRID is 4326 by default.

Examples

```
With traj AS (
  Select ST_makeTrajectory('STPOINT', 'LINESTRINGZ(114.000528 33.588163 54.87, 114.000535 33.588235 54.85, 114.000447 33.588272 54.69, 114.000348 33.588287 54.73, 114.000245 33.588305 55.26, 114.000153 33.588305 55.3)'::geometry,
                ARRAY['2010-01-01 11:30'::timestamp, '2010-01-01 11:31', '2010-01-01 11:32', '2010-01-01 11:33','2010-01-01 11:34','2010-01-01 11:35'], NULL) a,
      ST_makeTrajectory('STPOINT', 'LINESTRINGZ(114.000529 33.588163 54.87, 114.000535 33.578235 54.85, 114.000447 33.578272 54.69, 114.000348 33.578287 54.73, 114.000245 33.578305 55.26, 114.000163 33.588305 55.3)'::geometry,
                ARRAY['2010-01-01 11:29:58'::timestamp, '2010-01-01 11:31:02', '2010-01-01 11:33', '2010-01-01 11:33:09','2010-01-01 11:34','2010-01-01 11:34:30'], NULL) b)
Select st_LCSSimilarity(a, b, 100) from traj;
 st_lcssimilarity
------------------
 2
(1 row)
With traj AS (
  Select ST_makeTrajectory('STPOINT', 'LINESTRINGZ(114.000528 33.588163 54.87, 114.000535 33.588235 54.85, 114.000447 33.588272 54.69, 114.000348 33.588287 54.73, 114.000245 33.588305 55.26, 114.000153 33.588305 55.3)'::geometry,
                ARRAY['2010-01-01 11:30'::timestamp, '2010-01-01 11:31', '2010-01-01 11:32', '2010-01-01 11:33','2010-01-01 11:34','2010-01-01 11:35'], NULL) a,
      ST_makeTrajectory('STPOINT', 'LINESTRINGZ(114.000529 33.588163 54.87, 114.000535 33.578235 54.85, 114.000447 33.578272 54.69, 114.000348 33.578287 54.73, 114.000245 33.578305 55.26, 114.000163 33.588305 55.3)'::geometry,
                ARRAY['2010-01-01 11:29:58'::timestamp, '2010-01-01 11:31:02', '2010-01-01 11:33', '2010-01-01 11:34:15','2010-01-01 11:34:50','2010-01-01 11:34:30'], NULL) b)
Select st_LCSSimilarity(a, b, 100, interval '30 seconds') from traj;
 st_lcssimilarity
-------------------
2
(1 row)
```

7.16.2. ST_lcsDistance

This function computes the distance between two trajectory objects based on the longest common sub-sequence (LCSS) algorithm.

Syntax

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

Parameters

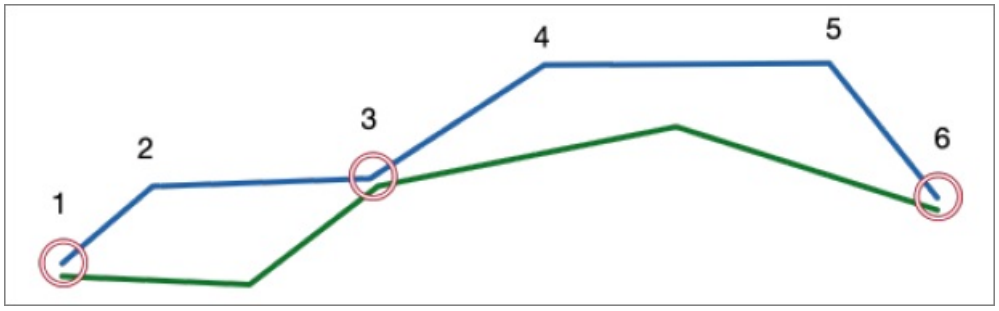
Parameter	Description
traj1	The trajectory object 1.
traj2	The trajectory object 2.

Parameter	Description
dist	The distance tolerance between two trajectory points. Unit: meters.
lag	The time tolerance between two trajectory points.
unit	The unit of the distance. Valid values: • 'M': meters. • 'KM': kilometers. • 'D': degrees. This value is valid only when the spatial reference system identifier (SRID) of a trajectory object is WGS84 (4326 by default).

Description

The LCSS algorithm is used to compute the maximum similarity between two trajectory objects. The consistency between two trajectory points is determined based on their distance in space and time.

This function computes the distance between two trajectory objects by using the following formula: 1 - (LCSS)/min(leafcount(traj1), leafcount(traj2)).



As shown in the preceding figure, trajectory points 1, 3, and 6 meet the conditions of consistency. In this case, the value of LCSS is 3, and 1 - 3/5 = 0.4. This function returns 0.4.

Generally, a trajectory object has a valid SRID. If not specified, the SRID is 4326 by default.

Examples

```
With traj AS (
  Select ST_makeTrajectory('STPOINT', 'LINESTRINGZ(114.000528 33.588163 54.87, 114.000535 33.588235 54.85, 114.000447 33.588272 54.69, 114.000348 33.588287 54.73, 114.000245 33.588305 55.26, 114.000153 33.588305 55.3)'::geometry,
                ARRAY['2010-01-01 11:30'::timestamp, '2010-01-01 11:31', '2010-01-01 11:32', '2010-01-01 11:33','2010-01-01 11:34','2010-01-01 11:35'], NULL) a,
      ST_makeTrajectory('STPOINT', 'LINESTRINGZ(114.000529 33.588163 54.87, 114.000535 33.578235 54.85, 114.000447 33.578272 54.69, 114.000348 33.578287 54.73, 114.000245 33.578305 55.26, 114.000163 33.588305 55.3)'::geometry,
                ARRAY['2010-01-01 11:29:58'::timestamp, '2010-01-01 11:31:02', '2010-01-01 11:33', '2010-01-01 11:33:09','2010-01-01 11:34','2010-01-01 11:34:30'], NULL) b)
Select st_LCSDistance(a, b, 100) from traj;
 st_lcsdistance
--------------------
 0.666666666666667
(1 row)
With traj AS (
  Select ST_makeTrajectory('STPOINT', 'LINESTRINGZ(114.000528 33.588163 54.87, 114.000535 33.588235 54.85, 114.000447 33.588272 54.69, 114.000348 33.588287 54.73, 114.000245 33.588305 55.26, 114.000153 33.588305 55.3)'::geometry,
                ARRAY['2010-01-01 11:30'::timestamp, '2010-01-01 11:31', '2010-01-01 11:32', '2010-01-01 11:33','2010-01-01 11:34','2010-01-01 11:35'], NULL) a,
      ST_makeTrajectory('STPOINT', 'LINESTRINGZ(114.000529 33.588163 54.87, 114.000535 33.578235 54.85, 114.000447 33.578272 54.69, 114.000348 33.578287 54.73, 114.000245 33.578305 55.26, 114.000163 33.588305 55.3)'::geometry,
                ARRAY['2010-01-01 11:29:58'::timestamp, '2010-01-01 11:31:02', '2010-01-01 11:33', '2010-01-01 11:34:15','2010-01-01 11:34:50','2010-01-01 11:34:30'], NULL) b)
Select st_LCSDistance(a, b, 100, interval '30 seconds') from traj;
 st_lcsdistance
--------------------
0.666666666666667
(1 row)
With traj AS (
  Select ST_makeTrajectory('STPOINT', 'LINESTRINGZ(114.000528 33.588163 54.87, 114.000535 33.588235 54.85, 114.000447 33.588272 54.69, 114.000348 33.588287 54.73, 114.000245 33.588305 55.26, 114.000153 33.588305 55.3)'::geometry,
                ARRAY['2010-01-01 11:30'::timestamp, '2010-01-01 11:31', '2010-01-01 11:32', '2010-01-01 11:33','2010-01-01 11:34','2010-01-01 11:35'], NULL) a,
      ST_makeTrajectory('STPOINT', 'LINESTRINGZ(114.000529 33.588163 54.87, 114.000535 33.578235 54.85, 114.000447 33.578272 54.69, 114.000348 33.578287 54.73, 114.000245 33.578305 55.26, 114.000163 33.588305 55.3)'::geometry,
                ARRAY['2010-01-01 11:29:58'::timestamp, '2010-01-01 11:31:02', '2010-01-01 11:33', '2010-01-01 11:34:15','2010-01-01 11:34:50','2010-01-01 11:34:30'], NULL) b)
Select st_LCSDistance(a, b, 100, interval '30 seconds', 'M') from traj;
 st_lcsdistance
--------------------
0.666666666666667
(1 row)
```

7.16.3. ST_lcsSubDistance

This function computes the distance between a longest common sub-sequence (LCSS) sub-trajectory and a trajectory object.

Syntax

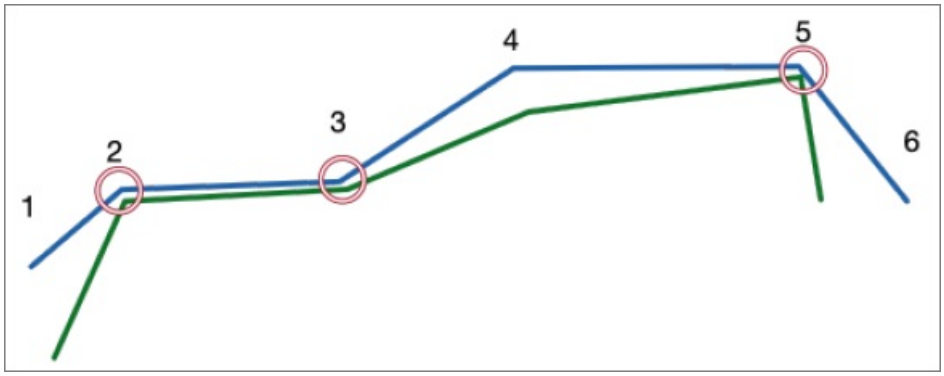
```
float8 ST_lcsSubDistance(trajectory traj1, trajectory traj2, float8 dist, distanceUnit unit default 'M');
float8 ST_lcsSubDistance(trajectory traj1, trajectory traj2, float8 dist, interval lag, distanceUnit unit default 'M');
```

Parameters

Parameter	Description
traj1	The trajectory object 1.
traj2	The trajectory object 2.
dist	The distance tolerance between two trajectory points. Unit: meters.
lag	The time tolerance between two trajectory points.
unit	The unit of the distance. Valid values: • 'M': meters. • 'KM': kilometers. • 'D': degrees. This value is valid only when the spatial reference system identifier (SRID) of a trajectory object is WGS84 (4326 by default).

Description

This function computes the sub-trajectory of trajectory object 1 in which trajectory points are consistent with those in the LCSS sub-trajectory, and then returns the ratio between the total number of trajectory points in the sub-trajectory of trajectory object 1 and the number of trajectory points in the LCSS sub-trajectory.



As shown in the preceding figure, trajectory points 2, 3, and 5 are in the LCSS sub-trajectory. Trajectory points 2, 3, 4, and 5 are in the sub-trajectory of trajectory object 1. In this case, 1 - 3/4 = 0.25. This function returns 0.25.

A smaller value indicates a higher similarity between the LCSS sub-trajectory and trajectory object 1.

Generally, a trajectory object has a valid SRID. If not specified, the SRID is 4326 by default.

Examples

```
With traj AS (
  Select ST_makeTrajectory('STPOINT', 'LINESTRINGZ(114.000528 33.588163 54.87, 114.000535 33.588235 54.85, 114.000447 33.588272 54.69, 114.000348 33.588287 54.73, 114.000245 33.588305 55.26, 114.000153 33.588305 55.3)'::geometry,
                ARRAY['2010-01-01 11:30'::timestamp, '2010-01-01 11:31', '2010-01-01 11:32', '2010-01-01 11:33','2010-01-01 11:34','2010-01-01 11:35'], NULL) a,
      ST_makeTrajectory('STPOINT', 'LINESTRINGZ(114.000529 33.588163 54.87, 114.000535 33.578235 54.85, 114.000447 33.578272 54.69, 114.000348 33.578287 54.73, 114.000245 33.578305 55.26, 114.000163 33.588305 55.3)'::geometry,
                ARRAY['2010-01-01 11:29:58'::timestamp, '2010-01-01 11:31:02', '2010-01-01 11:33', '2010-01-01 11:33:09','2010-01-01 11:34','2010-01-01 11:34:30'], NULL) b)
Select st_LCSSubDistance(a, b, 100) from traj;
st_lcssubdistance
------------------------
 0.66666666666666666667
(1 row)
 With traj AS (
  Select ST_makeTrajectory('STPOINT', 'LINESTRINGZ(114.000528 33.588163 54.87, 114.000535 33.588235 54.85, 114.000447 33.588272 54.69, 114.000348 33.588287 54.73, 114.000245 33.588305 55.26, 114.000153 33.588305 55.3)'::geometry,
                ARRAY['2010-01-01 11:30'::timestamp, '2010-01-01 11:31', '2010-01-01 11:32', '2010-01-01 11:33','2010-01-01 11:34','2010-01-01 11:35'], NULL) a,
      ST_makeTrajectory('STPOINT', 'LINESTRINGZ(114.000529 33.588163 54.87, 114.000535 33.578235 54.85, 114.000447 33.578272 54.69, 114.000348 33.578287 54.73, 114.000245 33.578305 55.26, 114.000163 33.588305 55.3)'::geometry,
                ARRAY['2010-01-01 11:29:58'::timestamp, '2010-01-01 11:31:02', '2010-01-01 11:33', '2010-01-01 11:33:09','2010-01-01 11:34','2010-01-01 11:34:30'], NULL) b)
Select st_LCSSubDistance(a, b, 100, interval '30 seconds') from traj;
st_lcssubdistance
-------------------
0.666666666666667
(1 row)
```

7.16.4. ST_JaccardSimilarity

This topic describes the ST_JaccardSimilarity function, which calculates the Jaccard index based on two trajectories or sub-trajectories.

Syntax

```
record ST_JaccardSimilarity(trajectory tr1, trajectory tr2, double tol_dist,
                text unit default '{}', interval tol_time default NULL,
                timestamp ts default '-infinity', timestamp te default 'infinity') ;
```

Parameters

Parameter	Description
tr1	The trajectory numbered 1.

Parameter	Description
tr2	The trajectory numbered 2.
tol_dist	The tolerance that is allowed for the distance between a pair of trajectory points. Unit: meters.
unit	The JSON string that specifies how to calculate the distance.
tol_time	The tolerance that is allowed for the time difference between a pair of trajectory points. The default value is NULL. If you set this parameter to NULL or a negative value, this function calculates the Jaccard index based only on the distance and does not consider the time difference.
ts	The start of the time range for the calculation. The default value is `-infinity`. If you specify this parameter, this function calculates only the sub-trajectories between the specified start time and end time.
te	The end of the time range for the calculation. The default value is `infinity`. If you specify this parameter, this function calculates only the sub-trajectories between the specified start time and end time.

The following table describes the fields in the unit parameter.

Field	Type	Default value	Description
Projection	string	None	The coordinate system to which the specified trajectories or sub-trajectories are re-projected. Valid values: • auto: This function dynamically selects the most suitable coordinate system based on the longitude and latitude coordinates of the specified trajectories or sub-trajectories. Then, this function re-projects the specified trajectories or sub-trajectories to the selected coordinate system. The supported coordinate systems include Lambert Azimuthal and UTM. You do not need to specify the Unit parameter again. The unit is meters. • srid: This function re-projects the specified trajectories or sub-trajectories based on the spatial reference identifier (SRID) that is specified by this parameter. **Note** If you do not specify this parameter, this function performs the calculation based on the original coordinate system.

Field	Type	Default value	Description
Unit	string	null	The unit that is used to measure the distance. Valid values: • null: No units are specified. This function calculates the Euclidean distance between each pair of trajectory points based on the coordinates of these trajectory points. • M: This function calculates the distance between each pair of trajectory points based on the unit that is used for the spatial reference of the specified trajectories or sub-trajectories. In most cases, the distance is measured in the unit of meters.
useSpheroid	bool	true	Specifies whether to use an ellipsoid. You can specify this parameter if you set the Unit parameter to M. Valid values: • true: An ellipsoid is used. This function can obtain accurate distances. • false: A sphere that represents the earth is used. This function can obtain approximate distances.

The following table describes the response parameters.

Parameter	Type	Description
nleaf1	int	The number of trajectory points where Trajectory 1 intersects with Trajectory 2.
nleaf2	int	The number of trajectory points where Trajectory 2 intersects with Trajectory 1. **Note** The return value of this parameter may differ from that of the nleaf1 parameter. Assume that Trajectory 1 passes the same trajectory point on Trajectory 2 twice. In this case, the return value of the nleaf1 parameter is 1, and the return value of the nleaf2 parameter is 2.
inter1	int	The number of trajectory points between which the distance from Trajectory 1 to Trajectory 2 meets both the specified time and distance tolerances.
inter2	int	The number of trajectory points between which the distance from Trajectory 2 to Trajectory 1 meets both the specified time and distance tolerances.

Parameter	Type	Description
jaccard_lower	double	The minimum distance that is calculated based on the Jaccard index. The value of this parameter is calculated based on the following formula: Minimum distance = `inter/(nleaf1 + nleaf2 - inter)`. **Note** In the preceding formula, the value of the inter parameter is the smaller value between the inter1 and inter2 parameters.
jaccard_upper	double	The maximum distance that is calculated based on the Jaccard index. The value of this parameter is calculated based on the following formula: Maximum distance = `inter/(nleaf1 + nleaf2 - inter)`. **Note** In the preceding formula, the value of the inter parameter is the larger one between the inter1 and inter2 parameters.

Description

The Jaccard index of two sets is the number of elements in the intersection of these sets divided by the number of elements in the union of these sets. For two trajectories, the definition of the Jaccard index is extended to calculate the number of trajectory points where Trajectory 1 intersects with Trajectory 2, the number of trajectory points where Trajectory 2 intersects with Trajectory 1, and the distance between each pair of trajectory points. The distances are calculated based on the preceding formulas that are used to calculate the values of the jaccard_lower and jaccard_upper parameters.

Example

```
With traj as(
SELECT ST_makeTrajectory('STPOINT'::leaftype, 'SRID=4326;LINESTRING(114.49211 37.97921,114.49211 37.97921,114.49211 37.97921,114.49211 37.97921)'::geometry,
  ARRAY[to_timestamp(1590287775) AT TIME ZONE 'UTC', to_timestamp(1590287778) AT TIME ZONE 'UTC', to_timestamp(1590302169) AT TIME ZONE 'UTC',to_timestamp(1590302171) AT TIME ZONE 'UTC'], '{}') a,
  ST_makeTrajectory('STPOINT'::leaftype, 'SRID=4326;LINESTRING(114.49211 37.97921,114.49211 37.97921,114.49211 37.97921,114.49211 37.97921,114.49145 37.97781,114.49145 37.97781,114.49145 37.97781,114.49145 37.97781,114.49145 37.97781,114.49145 37.97781,114.49145 37.97781,114.49145 37.97781,114.49145 37.97781,114.49211 37.97921,114.49211 37.97921,114.49211 37.97921,114.49211 37.97921,114.49211 37.97921,114.49211 37.97921,114.49211 37.97921)'::geometry,
  ARRAY[ to_timestamp(1590287765) AT TIME ZONE 'UTC', to_timestamp(1590287771) AT TIME ZONE 'UTC', to_timestamp(1590287778) AT TIME ZONE 'UTC', to_timestamp(1590287780) AT TIME ZONE 'UTC', to_timestamp(1590295992) AT TIME ZONE 'UTC', to_timestamp(1590295997) AT TIME ZONE 'UTC', to_timestamp(1590296013) AT TIME ZONE 'UTC', to_timestamp(1590296018) AT TIME ZONE 'UTC',  to_timestamp(1590296025) AT TIME ZONE 'UTC', to_timestamp(1590296032) AT TIME ZONE 'UTC', to_timestamp(1590296055) AT TIME ZONE 'UTC', to_timestamp(1590296073) AT TIME ZONE 'UTC', to_timestamp(1590296081) AT TIME ZONE 'UTC', to_timestamp(1590296081) AT TIME ZONE 'UTC', to_timestamp(1590302169) AT TIME ZONE 'UTC', to_timestamp(1590302174) AT TIME ZONE 'UTC',  to_timestamp(1590302176) AT TIME ZONE 'UTC', to_timestamp(1590302176) AT TIME ZONE 'UTC', to_timestamp(1590302172) AT TIME ZONE 'UTC', to_timestamp(1590302176) AT TIME ZONE 'UTC'],'{}') b
)
select ST_JaccardSimilarity(a,b, 100,'{"unit":"M"}','20 second', '2020-05-23'::timestamptz AT TIME ZONE 'UTC', '2020-05-26'::timestamptz AT TIME ZONE 'UTC') from traj;
    st_jaccardsimilarity
-----------------------------------
 (4,20,4,10,0.2,0.714285714285714)
(1 row)
```

7.17. Indexing

7.17.1. GiST indexing

This topic describes GiST indexing. You can create a GiST index on the column that stores trajectory data.

Syntax

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

- index_name: the name of the index. This parameter is optional.
- table_name: the name of the table to which the column belongs.
- traj_col: the name of the column.
- operator_family: the operator family that is used to create the index. Default value: trajgist_ops_3dt. This parameter is optional.

> **Note** The created index accelerates queries that are run by operators and the following functions: ST_ndIntersect, ST_ndDWithin, ST_ndContains, and ST_ndWithin.

Supported operator families

The following table describes the operator families that are supported by GiST.

Operator family	Description
trajgist_ops_z	Creates an index on the z axis. This type of index supports queries that cover only the z axis.
trajgist_ops_t	Creates an index on the t axis. This type of index supports queries that cover only the t axis.
trajgist_ops_2d	Creates an index on the x and y axes. This type of index supports queries that cover only the x and y axes.
trajgist_ops_2dt	Creates an index on the x, y, and t axes. This type of index supports queries that cover only the x and y axes, queries that cover only the t axis, and queries that cover the x, y, and t axes.
trajgist_ops_3d	Creates an index on the x, y, and z axes. This type of index supports queries that cover only the x and y axes, queries that cover only the z axis, and queries that cover the x, y, and z axes.
trajgist_ops_3dt	Creates an index on the x, y, z, and t axes. This type of index supports all of the queries that are supported by the preceding five operator families.

Note In most cases, a smaller number of dimensions in the created index indicates a higher speed per query. If you frequently run a specific type of query, we recommend that you specify another operator family that meets your dimension requirements in addition to the default trajgist_ops_3dt operator family.

Example

- Create an index on the t axis.

```
CREATE INDEX on table_name USING GIST(traj_col trajgist_ops_t);
```

- Create an index on the x and y axes.

```
CREATE INDEX on table_name USING GIST(traj_col trajgist_ops_2d);
```

- Create an index on the x, y, and t axes.

```
CREATE INDEX on table_name USING GIST(traj_col trajgist_ops_2dt);
```

7.17.2. TrajGiST indexing

This topic describes TrajGiST indexing, which is an extension to GiST indexing. You can create a TrajGiST index on a column that stores trajectory data.

Background information

TrajGiST performs better than GiST in the following ways:

- TrajGiST provides a better method that is used to estimate the overheads of indexes. If more than one index is created, TrajGiST can better select an index than GiST.
- TrajGiST supports upward compatibility of indexes. If an accurate result cannot be obtained because the index does not provide sufficient information, you can still obtain a coarse-grained result by using an index-based scan. For example, you have created an index only on the t axis, but you want to call the ST_{2DT}Intersects function. In this case, you can filter out the trajectories that do not fall within a specified time range based on the index on the t axis.

Syntax

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

- index_name: the name of the index. This parameter is optional.
- table_name: the name of the table to which the column belongs.
- traj_col: the name of the column.
- operator_family: the operator family that is used to create the index. Default value: trajgist_ops_3dt. This parameter is optional.

> **Note** The created index accelerates queries that are run by operators and the following functions: ST_ndIntersect, ST_ndDWithin, ST_ndContains, and ST_ndWithin.

Supported operator families

The following table describes the operator families that are supported by GiST.

Operator family	Description
trajgist_ops_z	Creates an index on the z axis. This type of index supports queries that cover only the z axis.
trajgist_ops_t	Creates an index on the t axis. This type of index supports queries that cover only the t axis.
trajgist_ops_2d	Creates an index on the x and y axes. This type of index supports queries that cover only the x and y axes.
trajgist_ops_2dt	Creates an index on the x, y, and t axes. This type of index supports queries that cover the x, y, and t axes.
trajgist_ops_3d	Creates an index on the x, y, and z axes. This type of index supports queries that cover the x, y, and z axes.
trajgist_ops_3dt	Creates an index on the x, y, z, and t axes. This type of index supports all of the queries that are supported by the preceding five operator families.

> **Note** TrajGiST allows you to create an index on a single column. It does not allow you to create an index on multiple columns including the columns that store other types of data than trajectory data.

7.18. External storage

7.18.1. ST_ExportTo

You can use this function to export the data of a trajectory object to a folder in an Object Storage Service (OSS) bucket.

Syntax

```
trajectory ST_ExportTo(trajectory traj, text path, text config);
```

Parameters

Parameter	Description
traj	The trajectory object.
path	The string that specifies the folder in which you want to store the data of the specified trajectory object. Only folders in OSS are supported. The format of the parameter value is 'OSS://<AccessKey ID>:< AccessKey secret>@<Endpoint>/<Bucket>/<Directory>'. The data of the specified trajectory object is stored in a .gtf file in the specified folder.
config	A JSON string. The following list describes the parameter in this string: • Parameter: compress • Default value: none • Valid values: none, lz4, lzo, zstd, snappy, and zlib

Description

This function stores the time, space, event, and property information of a specified trajectory object to the specified folder in OSS.

Replace <Endpoint> with the endpoint that you use to access OSS in the region where the trajectory data is stored. To ensure import performance, make sure that the PolarDB O Edition database is deployed in the same region as the OSS bucket. For more information, see OSS domain names.

The binary data of the trajectory object is appended to a *.gtf* file in the specified folder. The format of the .gtf file name is *<Sequence number of the file>_0_1.gtf*. The .gtf files in a folder are sorted in descending order by sequence number. The size of a single file is approximately 2 GB. The size is specified by the ganos.trajectory.ext_storage_block_size parameter.

If a *.gtf* file that stores the data of the specified trajectory object already exists in the specified folder, the system appends the data of the trajectory object to the existing *.gtf* file. If the data of the specified trajectory object is stored in a folder in OSS and the system detects that the trajectory data is already stored in the specified folder, the system does not write the data to the folder again. If the system detects that the trajectory data is not stored in the specified folder, the system reads the data and then writes the data to the folder.

If you delete or update the data in the database that stores the trajectory data, the data in the OSS file is not updated. If you update the data in the OSS file, the data in the database can become unavailable.

You can choose a compression algorithm based on your business requirements. The zlib algorithm provides the highest compression ratio. The zstd algorithm provides the highest overall efficiency.

Examples

```
update trajs
set traj = ST_exportTo(traj, 'OSS://<access key>:<access secret>@oss-cn-beijing-internal/<bucket>/<directory>', '{}');
UPDATE 113395
```

7.18.2. ST_IsExternal

You can use this function to check whether the data of a trajectory object is stored in Object Storage Service (OSS).

Syntax

```
bool ST_IsExternal(trajectory traj);
```

Parameters

Parameter	Description
traj	The trajectory object.

Description

If the data of the specified trajectory object is stored in OSS, the function returns true. Otherwise, this function returns false.

Examples

```
select ST_IsExternal(traj) from trajs;
 st_isexternal
---------------
 t
 t
 t
 f
 t
(5 rows)
```

7.18.3. ST_importFrom

You can use this function to import the data of a trajectory object from Object Storage Service (OSS) to the PolarDB O Edition database that is used to store the trajectory data.

Syntax

```
trajectory ST_importFrom(trajectory traj);
```

Parameters

Parameter	Description
traj	The trajectory object.

Description

This function performs operations that are the reverse of the operations performed by the ST_ExportTo function. If the data of the specified trajectory object is stored in a PolarDB O Edition database table, this function returns the trajectory data. If the trajectory data is stored in a file in OSS, this function imports the trajectory data into the PolarDB O Edition database that is used to store the trajectory data. You can use this function and the `Update` command to change the data storage location for the specified trajectory object from OSS to the PolarDB O Edition database that is used to store the trajectory data.

This function does not change the OSS file that stores the trajectory data.

Examples

```
update trajs
set traj = ST_ImportFrom(traj);
UPDATE 4
```

7.18.4. ST_StorageLocation

You can use this function to query the location of the Object Storage Service (OSS) file that stores the data of a trajectory object.

Syntax

```
bool ST_StorageLocation(trajectory traj);
```

Parameters

Parameter	Description
traj	The trajectory object.

Description

If the data of the specified trajectory object is stored in a *.gtf* file in OSS, this function returns the information of the folder that stores the file. Otherwise, this function returns null.

Examples

```
select ST_StorageLocation(traj) from trajs;
                                st_storagelocation
-------------------------------------------------------------------------------------------------------------------------
 OSS://$(OSS_USER):<AccessKeySecret>@$(OSS_ENDPOINT)/$(OSS_BUCKET)/$(OSS_E_EXPORT)1/
 OSS://$(OSS_USER):<AccessKeySecret>@$(OSS_ENDPOINT)/$(OSS_BUCKET)/$(OSS_E_EXPORT)2/
 OSS://$(OSS_USER):<AccessKeySecret>@$(OSS_ENDPOINT)/$(OSS_BUCKET)/$(OSS_E_EXPORT)2/
(4 rows)
```

7.18.5. ST_AKID

You can use this function to query the AccessKey ID that is used to access the Object Storage Service (OSS) bucket in which the data of a trajectory object is stored.

Syntax

```
text ST_AKID(trajectory traj);
```

Parameters

Parameter	Description
traj	The trajectory object.

Description

If the data of a specified trajectory is stored in an OSS bucket, this function returns the AccessKey ID that is used to access the bucket. Otherwise, this function returns null.

When you read trajectory data from a file in an OSS bucket, you must provide the AccessKey pair information to access the OSS bucket. An AccessKey pair consists of an AccessKey ID and an AccessKey secret. An AccessKey ID is similar to a username and an AccessKey secret is similar to a password.

Examples

```
select ST_AKID(traj) from trajs;
     st_akid
---------------------------
 $(OSS_USER)
 $(OSS_USER)
 $(OSS_USER)
```

7.18.6. ST_SetAccessKey

You can invoke this function to set an AccessKey pair. The AccessKey pair is used to access the Object Storage Service (OSS) bucket that stores the data of a trajectory object.

Syntax

```
text ST_SetAccessKey(trajectory traj, text akid, text aksecret, bool checkvalid default true);
```

Parameters

Parameter	Description
traj	The trajectory object.
akid	The AccessKey ID that is used to access the OSS bucket.
aksecret	The AccessKey secret that is used to access the OSS bucket.
checkvalid	Specifies whether to check the validity of the AccessKey pair. Default value: true. If this parameter is set to true, the system tests whether the specified AccessKey pair can be used to access the OSS bucket. If the access test fails, an error is returned.

Description

This function resets an AccessKey pair. The AccessKey pair is used to access the OSS bucket that stores the data of a specified trajectory object.

When you read trajectory data from a file in an OSS bucket, you must provide the AccessKey pair information to access the OSS bucket. An AccessKey pair consists of an AccessKey ID and an AccessKey secret. An AccessKey ID is similar to a username and an AccessKey secret is similar to a password.

Examples

```
select ST_SetAccessKey(traj, '<OSS_USER>', '<OSS_PWD>') from trajs;
                                                                                                        st_
setaccesskey

------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------
------------------------------------------------
 null
 null
 null
 TRAJECTORY EMPTY
(4 rows)
```

7.18.7. ST_SetAkId

You can use this function to set an AccessKey ID. The AccessKey ID is used to access the Object Storage Service (OSS) bucket that stores data of a trajectory object.

Syntax

```
text ST_SetAkId(trajectory traj, text akid, bool checkvalid default true);
```

Parameters

Parameter	Description
traj	The trajectory object.
akid	The AccessKey ID that is used to access the OSS bucket.
checkvalid	Specifies whether to check the validity of the AccessKey ID. Default value: true. If this parameter is set to true, the system tests whether the specified AccessKey ID can be used to access the OSS bucket. If the access test fails, an error is returned.

Description

This function resets an AccessKey ID. The AccessKey ID is used to access the OSS bucket that stores the data of a specified trajectory object.

When you read trajectory data from a file in an OSS bucket, you must provide the AccessKey pair information to access the OSS bucket. An AccessKey pair consists of an AccessKey ID and an AccessKey secret. An AccessKey ID is similar to a username and an AccessKey secret is similar to a password.

Examples

```
select ST_SetAKId(traj, '<OSS_USER>') from trajs;
                                                                                                    st
_setakid

-----------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------
 null
 null
 null
 TRAJECTORY EMPTY
(4 rows)
```

7.18.8. ST_SetAkSecret

You can use this function to set an AccessKey secret. The AccessKey secret is used to access the Object Storage Service (OSS) bucket that stores the data of a specified trajectory object.

Syntax

```
text ST_SetAkSecret(trajectory traj, text aksecret, bool checkvalid default true);
```

Parameters

Parameter	Description
traj	The trajectory object.

Parameter	Description
aksecret	The AccessKey secret that is used to access the OSS bucket.
checkvalid	Specifies whether to check the validity of the AccessKey secret. Default value: true. If this parameter is set to true, the system tests whether the specified AccessKey secret can be used to access the OSS bucket. If the access test fails, an error is returned.

Description

This function resets an AccessKey secret. The AccessKey secret is used to access the OSS bucket that stores the data of a specified trajectory.

When you read trajectory data from a file in an OSS bucket, you must provide the AccessKey pair information to access the OSS bucket. An AccessKey pair consists of an AccessKey ID and an AccessKey secret. An AccessKey ID is similar to a username and an AccessKey secret is similar to a password.

Examples

```
select ST_SetAkSecret(traj, '<OSS_PWD>') from trajs;
                                                                                                   st_
setaksecret

-----------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------
 null
 null
 null
 TRAJECTORY EMPTY
(4 rows)
```

7.18.9. ST_SetStorageLocation

You can use this function to change the location of the Object Storage Service (OSS) file that stores the data of a trajectory object.

Syntax

```
bool ST_SetStorageLocation(trajectory traj, text new_path, bool checkValid default true);
```

Parameters

Parameter	Description
traj	The trajectory object.

Parameter	Description
new_path	The string that specifies the folder in which you want to store the data of the specified trajectory object. Only folders in OSS are supported. The format of the parameter value is 'OSS://<AccessKey ID>:< AccessKey secret>@<Endpoint>/<Bucket>/<Directory>'. The data of the specified trajectory object is stored in a *.gtf* file in the specified folder.
checkvalid	Specifies whether to check the validity of the path. Default value: true. If this parameter is set to true, the system tests whether the specified OSS directory can be accessed by using your AccessKey pair. If the access test fails, an error is returned.

Description

This function changes the location of the file that stores the data of a specified trajectory object. This function does not move the original file that stores the data of the trajectory object. Before you use this function to specify a new location, you first need to move the folder that contains the trajectory data file in OSS to the new location.

After the file is moved to the new location, new trajectory data may not be appended to the file. In this case, an error message that contains `is not appendable` is returned. Make sure that the last data block is uploaded by calling the AppendObject operation of OSS.

Examples

```
select ST_SetStorageLocation(traj, '<New Path>') from trajs;
                                                                                                                                st_s
etstoragelocation

-----------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------
 null
 null
 null
 TRAJECTORY EMPTY
(4 rows)
```

7.18.10. ST_DeleteGTF

You can use this function to delete all the trajectory data files from an Object Storage Service (OSS) folder.

Syntax

```
void ST_DeleteGTF(text directory);
```

Parameters

Parameter	Description
directory	The string that specifies the folder in which trajectory data files are located. Only folders in OSS are supported. The format of the parameter value is 'OSS://<AccessKey ID>:< AccessKey secret>@<Endpoint>/<Bucket>/<Directory>'. Trajectory data is stored in *.gtf* files in the specified folder.

Description

This function deletes the files that store trajectory data.

Examples

```
select ST_deleteGtf('OSS_PATH');
 st_deletegtf
--------------
(1 row)
```

7.19. Variables

7.19.1. ganos.trajectory.attr_string_length

This variable sets a default length for string-type attribute fields.

Description

The value of the `attr_string_length` parameter is integer.

Examples

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

8.GeomGrid SQL reference

8.1. Usage notes

GeomGrid is the data type that is used to represent grids in the Ganos GeomGrid extension.

Data types

- The following data types can be converted into the GeomGrid data type:
  - text
  - bytea
- The GeomGrid data type can be converted into the following data types:
  - text
  - bytea
  - geometry
  - box

8.2. Functions for output

8.2.1. ST_AsText

This function converts a grid object into text that is encoded in a specified specification.

Syntax

```
text ST_AsText(geomgrid   grid,
        integer precision,
        text standard default 'GGER')
text[] ST_AsText(geomgrid[]   grid,
        integer precision,
        text standard default 'GGER')
```

Parameters

Parameter	Description
grid	The grid object to be generated.
precision	The precision level. Valid values: 0 to 31. The value -1 indicates that the default precision level is used.
standard	The specification standard: GeoSpatial Grid Encoding Rule (GGER). It is developed by the Ministry of Natural Resources of the People's Republic of China. Default value: GGER.

Description

The grid object is generated based on the specified layer, precision, and encoding specification. If the specified precision is higher than the precision of the stored grid, the system does not fill zeros for output.

Examples

```
--Use the default layer.
with g as (
 select unnest(st_asgrid(
  ST_geomfromtext('POINT(116.31522216796875 39.910277777777778)',4490), 15)) as grid)
select ST_asText(grid) from g;
  st_astext
------------------
 G001310322230230
--Specify the output layer.
with g as (
 select unnest(st_asgrid(
  ST_geomfromtext('POINT(116.31522216796875 39.910277777777778)',4490), 15)) as grid)
select ST_asText(grid, 8) from g;
 st_astext
-------------
 G01310322
```

8.2.2. ST_AsBinary

This topic describes the ST_AsBinary function, which converts a grid into a binary structure.

Syntax

```
bytea ST_AsBinary(geomgrid  grid)
```

Parameters

Parameter	Description
grid	The grid that you want to convert.

Examples

```
with g as (
select unnest(st_asgridcode(
  ST_geomfromtext('POINT(116.31522216796875 39.910277777777778)',4490), 15)) as grid)
select ST_asBinary(grid) from g;
  st_asbinary
---------------------
 \x01010c0f74271236
```

8.2.3. ST_AsGeometry

This topic describes the ST_AsGeometry function, which obtains the geometry-represented range of a grid.

Syntax

```
geometry ST_AsGeometry(geomgrid grid);
geometry[] ST_AsGeometry(geomgrid[] grid);
```

Parameters

Parameter	Description
grid	The grid whose range you want to obtain.

Examples

```
select st_astext(
  st_asgeometry(st_gridfromtext('G0013103220310313')));
    st_astext
-------------------------------------------------------------------------------
 POLYGON((116.458888888889 39.3088888888889,116.458888888889 39.3166666666667,11
6.466666666667 39.3166666666667,116.466666666667 39.3088888888889,116.4588888888
89 39.3088888888889))
```

8.2.4. ST_AsBox

This topic describes the ST_AsBox function, which obtains the box-represented range of a grid.

Syntax

```
box ST_AsBox(geomgrid grid);
box[] ST_AsBox(geomgrid[] grid);
```

Parameters

Parameter	Description
grid	The grid whose range you want to obtain.

Examples

```
select st_asbox(
  st_gridfromtext('G0013103220310313'));
                    st_asbox
-------------------------------------------------------------------------------
(116.46666666666667,39.31666666666667),(116.4588888888889,39.30888888888889)
```

8.3. Functions for input

8.3.1. ST_GridFromText

This topic describes the ST_GridFromText function, which converts a grid from a string into a GeomGrid structure.

Syntax

```
geomgrid ST_GridFromText(text  Code)
```

Parameters

Parameter	Description
Code	The string of the grid that you want to convert.

Examples

```
select st_gridfromtext('G0013103220310313');
  st_gridfromtext
----------------------
 01011C1074271B120101
```

8.3.2. ST_GridFromBinary

This topic describes the ST_GridFromBinary function, which converts a grid from a binary structure into a GeomGrid structure.

Syntax

```
geomgrid ST_GridFromBinary(bytea  binary)
```

Parameters

Parameter	Description
binary	The binary structure of the grid that you want to convert.

Examples

```
select st_astext(
 st_gridfrombinary('\x01010c0f74271236'));
  st_astext
------------------
 G001310322230230
```

8.4. Functions to query spatial relationships

8.4.1. ST_Intersects

This topic describes the ST_Intersects function, which queries whether a grid intersects with a geometry.

Syntax

```
boolean ST_Intersects(geomgrid grid, geometry geom);
boolean ST_Intersects(geometry geom, geomgrid grid);
```

Parameters

Parameter	Description
grid	The grid whose spatial relationship you want to query.
geom	The geometry whose spatial relationship you want to query.

Description

This function returns the spatial relationship between a grid and a geometry. The geometry must use the CGC2000 spatial reference system. In addition, the spatial reference system identifier (SRID) of the geometry must be 4490.

Examples

```
select st_intersects(ST_gridfromtext('G001331032213300013'),
    ST_geomfromtext('LINESTRING(122.48077 51.72814, 122.47416 51.73714)',4490));
 st_intersects
---------------
 t
```

8.4.2. ST_Contains

This topic describes the ST_Contains function, which queries whether a grid contains a geometry.

Syntax

```
boolean ST_Contains(geomgrid grid, geometry geom);
boolean ST_Contains(geometry geom, geomgrid grid);
boolean ST_Contains(geomgrid grid1, geomgrid grid2);
```

Parameters

Parameter	Description
grid	The grid whose spatial relationship you want to query.
geom	The geometry whose spatial relationship you want to query.

Examples

```
select st_contains(ST_gridfromtext('G001331032213300013'),
    ST_geomfromtext('POINT(116.31522216796875 39.910277777777778)',4490));
 st_intersects
---------------
 f
 select st_contains(ST_gridfromtext('G00133103221330'),
             ST_gridfromtext('G001331032213300013'))
  st_contains
-------------
 t
```

8.4.3. ST_Within

This topic describes the ST_Within function, which queries whether a grid is contained in a geometry.

Syntax

```
boolean ST_Within(geomgrid grid, geometry geom);
boolean ST_Within(geometry geom, geomgrid grid);
boolean ST_Within(geomgrid grid1, geomgrid grid2);
```

Parameters

Parameter	Description
grid	The grid whose spatial relationship you want to query.
geom	The geometry whose spatial relationship you want to query.

Description

This function returns the spatial relationship between a grid and a geometry. The geometry must use the CGC2000 spatial reference system. In addition, the spatial reference system identifier (SRID) of the geometry must be 4490.

Examples

```
select st_within(ST_geomfromtext('POINT(116.31522216796875 39.910277777777778)',4490),
 ST_gridfromtext('G001331032213300013'));
 st_within
---------------
f
 select st_within(ST_gridfromtext('G001331032213300013'),
          ST_gridfromtext('G001331032213300'))
 st_within
-------------
t
```

8.5. Operators

8.5.1. @>

This topic describes the @> operator, which queries whether a grid contains a geometry.

Syntax

```
boolean @>(geomgrid grid, geometry geom);
boolean @>(geometry geom, geomgrid grid);
boolean @>(geomgrid grid1, geomgrid grid2);
```

Parameters

Parameter	Description
grid	The grid whose spatial relationship you want to query.
geom	The geometry whose spatial relationship you want to query.

Description

This function returns the spatial relationship between a grid and a geometry. The geometry must use the CGC2000 spatial reference system. In addition, the spatial reference system identifier (SRID) of the geometry must be 4490.

Examples

```
select ST_gridfromtext('G001331032213300013') @>
    ST_geomfromtext('POINT(116.31522216796875 39.910277777777778)',4490) as t;
t
---------------
f
select ST_gridfromtext('G00133103221330') @>
    ST_gridfromtext('G001331032213300013') as t
t
-------------
t
```

8.5.2. <@

This topic describes the <@ operator, which queries whether a grid is contained in a geometry.

Syntax

```
boolean <@(geomgrid grid, geometry geom);
boolean <@(geometry geom, geomgrid grid);
boolean <@(geomgrid grid1, geomgrid grid2);
```

Parameters

Parameter	Description
grid	The grid whose spatial relationship you want to query.
geom	The geometry whose spatial relationship you want to query.

Description

This function returns the spatial relationship between a grid and a geometry. The geometry must use the CGC2000 spatial reference system. In addition, the spatial reference system identifier (SRID) of the geometry must be 4490.

Examples

```
select ST_geomfromtext('POINT(116.31522216796875 39.910277777777778)',4490) <@
 ST_gridfromtext('G001331032213300013') as t;
t
---------------
f
select ST_gridfromtext('G001331032213300013') <@
    ST_gridfromtext('G001331032213300') as t
t
-------------
t
```

8.6. Functions to compute grids

8.6.1. ST_AsGrid

This topic describes the ST_AsGrid function, which queries the grids that intersect with a geometry.

Syntax

```
geomgrid[] ST_AsGrid(geometry geom, integer precision);
```

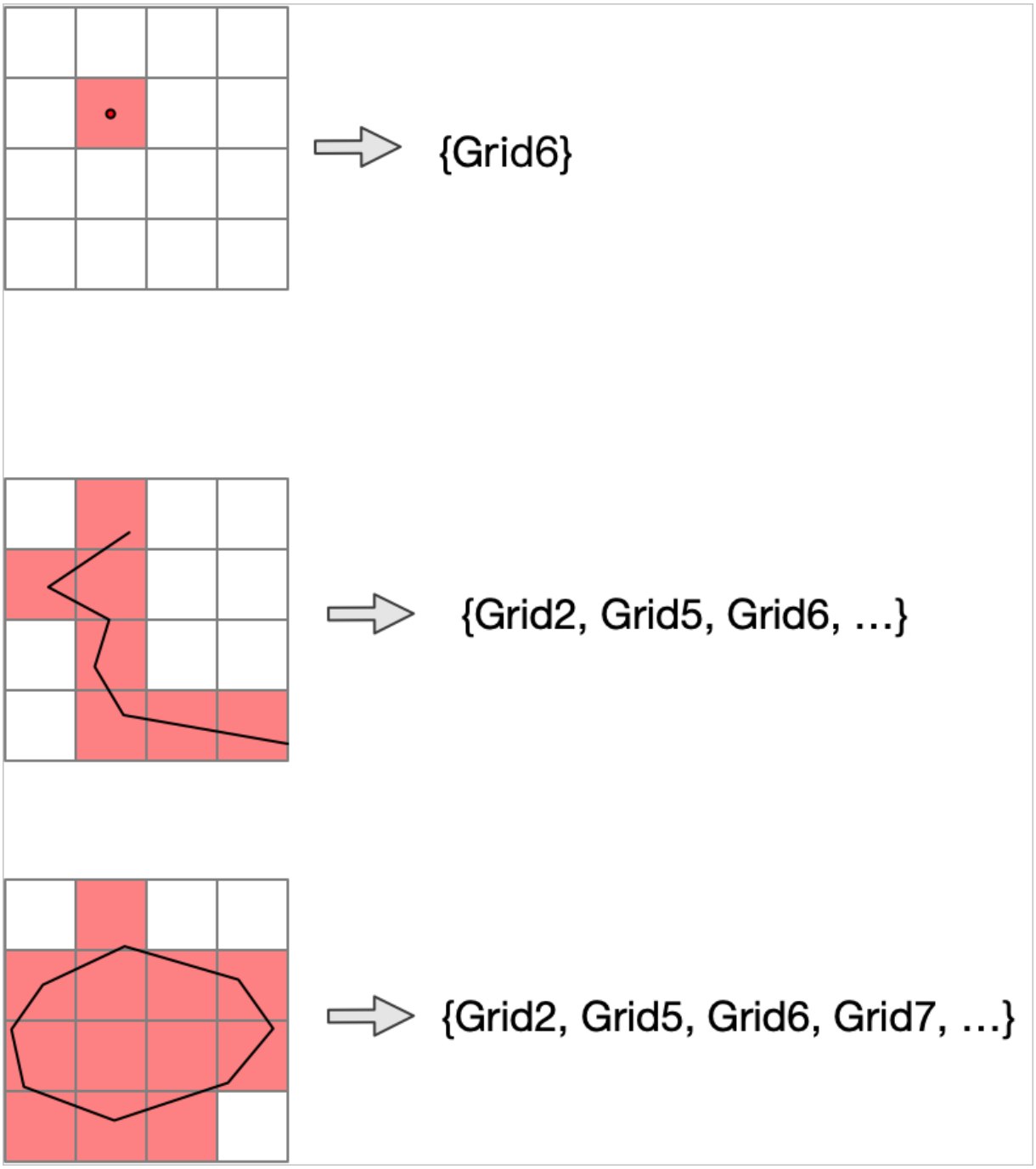
Parameters

Parameter	Description
geom	The geometry that you want to query.
precision	The precision level based on which you want to query the geometry. Valid values: 0 to 31.

Description

The geometry must use the CGC2000 spatial reference system. In addition, the spatial reference system identifier (SRID) of the geometry must be 4490. If the geometry does not use the CGC2000 spatial reference system, the ST_Transform function is invoked to convert the coordinates of the geometry into CGC2000 coordinates.

This function returns an array that consists of geometry-represented grids that intersect. The following figures are examples of how grids intersect with a point, a line, and a polygon.



Examples

```
select st_astext(st_asgrid(
  ST_geomfromtext('POINT(116.31522216796875 39.910277777777778)',4490), 15))
   st_astext
--------------------
 {G001310322230230}
select st_astext(st_asgrid(
  ST_geomfromtext('LINESTRING(122.48077 51.72814,122.47416 51.73714)',4490), 18))
           st_astext
-----------------------------------------------------------------------------------
 {G001331032213300011,G001331032213300013,G001331032213122320,G00133103221312232
2,G001331032213300100,G001331032213122303,G001331032213122321,G00133103221312231
2}
```

9.Geometry Pyramid SQL reference

9.1. Usage notes

Ganos Geometry Pyramid is used to display two-dimensional geometry big data at high speeds.

Data requirements

- Data tables can contain only the following types of spatial data: Point, Line, Polygon, MultiPoint, MultiLine, and MultiPolygon.
- Each data record must contain an ID field whose value is a unique positive integer of the uint4 or uint8 data type.
- You must create spatial indexes on the geometry fields.
- Geometry data supports all `EPSG` coordinate systems. MTP-formatted geometry data supports only the `EPSG:3857 and EPSG:4326` coordinate systems.

Hardware requirements

- We recommended that you configure 32 cores, 64 GB of memory, and 500 GB of SSD storage capacity. This allows you to process 0.1 billion data records within about 15 minutes.
- In most cases, the extra storage space that is occupied by the produced pyramids does not exceed 10% of the storage space that is occupied by the original data.

9.2. Functions to build pyramids

9.2.1. ST_BuildPyramid

This topic describes the ST_BuildPyramid function, which creates a vector pyramid for a spatial geometry data table to accelerate the display of data.

Syntax

```
boolean ST_BuildPyramid(cstring table, cstring geom, cstring fid, cstring config)
```

Parameters

Parameter	Description
table	The name of the spatial geometry data table.
geom	The name of the geometry field.
fid	The name of the element ID field.
config	The fields that are used to create a pyramid. These fields are specified as a JSON string.

The following table describes the fields in the config parameter.

Field	Type	Default value	Description
name	string	Same as the table name	The name of the pyramid.
parallel	int	0	The maximum number of tasks that can run in parallel to create the pyramid. You must also specify the max_prepared_transactions parameter. If you set the parallel parameter to 0, the maximum number of parallel tasks is not limited.
tileSize	int	1024	The tile size of the pyramid. Valid values must be greater than 0 and less than 4096.
tileExtend	int	8	The tile extension size of the pyramid. Valid values must be greater than 0.
userExtent	array[double]	null	The geographic bound that you define. The value of this parameter is a JSON string. This JSON string consists of four fields: minx, miny, maxx, and maxy. If you set this parameter to [], the value is null.
splitSize	int	10000	The maximum number of elements into which an index node can be split. A larger value of this parameter indicates a sparser pyramid.
maxLevel	int	16	The maximum number of layers in the pyramid. Valid values: 0 to 20.
sourceSRS	int	-1	The coordinate system of the source data in the spatial geometry data table. If you do not specify this parameter, the system reads the spatial reference identifiers (SRIDs) in the metadata.
destSRS	int	3857	The EPSG code of the title coordinate system. Only EPSG 3857 and EPSG 4326 are supported.
buildRules	array[object]	null	The rule based on which you want to create the pyramid. You can specify multiple rules. Each rule consists of two parts: level and value.
└level	array[int]	None	The pyramid layers to which the specified rule is applied.
└value	object	None	The value of the specified rule.

Field	Type	Default value	Description
└filter	string	None	The expression that is used to filter data in PostgreSQL.
└attrFields	array[string]	None	The name of the attribute field when the MVT format is used.
└merge	array[string]	None	The filter conditions that are used to add data records to a group.

The following example shows how to set the config parameter:

```
{
  "name" : "hello",
  "parallel": 4,
  "tileSize": 512,
  "tileExtend": 8,
  "userExtent": [-180,-90,180,90],
  "splitSize": 5000,
  "maxLevel": 16,
  "destSRS": 3857,
  "buildRules": [
   {
    "level": [0,1,2],
    "value": {
     "filter": "code!=0",
     "attrFields": ["name","color"],
     "merge":["code=1"]
    }
   }
  ]
}
```

Examples

```
--Create a pyramid for a spatial geometric data table named roads.
select ST_BuildPyramid('roads', 'geom', 'id', '');
st_buildpyramid
----------
t
```

9.3. Functions to delete pyramids

9.3.1. ST_DeletePyramid

This topic describes the ST_DeletePyramid function, which deletes a vector pyramid.

Syntax

```
boolean ST_DeletePyramid(cstring name)
```

Parameters

Parameter	Description
name	The name of the pyramid that you want to delete.

Examples

```
--Delete a pyramid.
select ST_DeletePyramid('roads');
st_deletepyramid
----------
t
```

9.4. Functions to view pyramids

9.4.1. ST_Tile

This topic describes the ST_Tile function, which generates a MVT-formatted standard binary structure based on the ID of a tile from a pyramid.

Syntax

```
bytea ST_Tile(cstring name, cstring key);
bytea ST_Tile(cstring name, int x, int y, int z);
```

Parameters

Parameter	Description
name	The name of the pyramid.
key	The ID of the tile.
x	The x value in the tile ID.
y	The y value in the tile ID.
z	The z value in the tile ID.

Description

This function returns a MVT-formatted standard binary structure that is generated based on the ID of a tile from a pyramid. The tile ID that is specified by the key parameter is encoded in the `'z_x_y'` format based on the `EPSG:4326` or `EPSG:3857` coordinate system.

Note If the `EPSG:4326` coordinate system is used, the number of chunks on the x axis is twice the number of chunks on the y axis. The z value starts from 1. If the z value is 1, the tile ID can only be `1_0_0` or `1_1_0`.

Example

```
select ST_Tile('roads', '3_1_6');
st_tile
----------
0xFFAABB8D8A6678...
select ST_Tile('roads', 1, 6, 3);
st_tile
----------
0xFFAABB8D8A6678...
```

9.4.2. ST_AsPng

This topic describes the ST_AsPng function, which converts a specific tile of a pyramid to a PNG image based on the ID of the tile.

Syntax

```
bytea ST_AsPng(cstring name, cstring key, cstring style);
bytea ST_AsPng(cstring name, int x, int y, int z, cstring style);
```

Parameters

Parameter	Description
name	The name of the pyramid.
key	The ID of the tile.
x	The x coordinate in the ID of the tile.
y	The y coordinate in the ID of the tile.
z	The z coordinate in the ID of the tile.
style	The style that is used to render the PNG image. The value of this parameter is a JSON string.

The following table describes the fields in the style parameter.

Field	Type	Default value	Description
background	string	#FFFFFFFF	The RGBA color of the background in the PNG image. The default color is white.

Field	Type	Default value	Description
line_color	string	#000000FF	The RGBA color of dots and edges in the PNG image. The default color is black.
fill_color	string	#F4A460FF	The RGBA color of the tile in the PNG image. The default color is brown.
line_width	int	1	The width per line in the PNG image. Unit: pixels.
point_size	int	10	The size per dot in the PNG image. Unit: pixels. By default, each dot is a circle with a diameter of 10 pixels.
parallel_unit	int	50000	The number of elements that each task running in parallel renders.

The following example shows the settings of the fields in the style parameter:

```
{
 "background": "#FFFFFFFF",
 "line_color": "#000000FF",
 "fill_color": "#F4A460FF",
 "line_width": 1,
 "point_size": 10
}
```

Description

The tile ID that is specified by the key parameter is in the `'z_x_y'` format in compliance with the `EPSG:4326` or `EPSG:3857` coordinate system.

> **Note**
>
> - A title consists of multiple chunks. If the `EPSG:4326` coordinate system is used, the number of chunks on the x axis is twice the number of chunks on the y axis. The z coordinate starts from 1. If the z coordinate is 1, the tile ID can only be `1_0_0` or `1_1_0`.
> - The size of the PNG image that is rendered based on the style parameter is equal to the value of the tileSize parameter that is specified for the pyramid. If you do not specify the style parameter, the default style is used.

Examples

```
select ST_AsPng('roads', '3_1_6', '');
st_aspng
----------
0xFFAABB8D8A6678...
select ST_AsPng('roads', 1, 6, 3, '');
st_aspng
----------
0xFFAABB8D8A6678...
```

10.Service publishing

10.1. Overview

GeoServer is a Java 2 Platform, Enterprise Edition (J2EE)-based software server in compliance with OpenGIS standards. It allows you to publish, update, delete, and insert geospatial data. You can share spatial information by using GeoServer.

GeoServer conforms to both the Web Map Service (WMS) and Web Feature Service (WFS) standards set forth by Open Geospatial Consortium (OGC). It supports PostgreSQL, Shapefile, ArcSDE, Oracle, VPF, MySQL, MapInfo, and projections in hundreds of formats. GeoServer can produce maps in various formats, such as JPEG, GIF, PNG, SVG, and KML. GeoServer can run in any Java-based or Servlet-based container. It is integrated with OpenLayers, a mapping library that allows you to use asynchronous JavaScript and XML (AJAX) requests to create maps in Map Builder. Additional features are provided for you to explore.

Download and install GeoServer

- GeoServer 2.13.X and GeoTools 19.X
- GeoServer 2.14.X and GeoTools 20.X
- GeoServer 2.15.X and GeoTools 21.X
- GeoServer 2.16.X and GeoTools 22.X
- GeoServer 2.17.X and GeoTools 23.X

> **Note** We recommend that you download GeoServer 2.14.2 or later.

Decompress the GeoServer package to the geoserver/WEB-INFO/lib directory and then start the container.

10.2. Publish geometry data

This topic describes how to publish geometry data in a Ganos database by using GeoServer.

Add a data store

1. Run GeoServer. In the left-side navigation pane, click **Stores**.
2. On the Stores page, click **Add new Store**. On the page that appears, select **PostGIS** in the Vector Data Sources section.
3. Set the connection information about the database.

Use the RESTful API

After publishing geometry data, you can use the RESTful API provided by GeoServer to access the data. For more formation, see GeoServer User Manual.

10.3. Publish raster data

This topic describes how to publish raster data in a Ganos database by using GeoServer.

Add a data source

1. Run GeoServer. In the left-side navigation pane, click **Stores**.
2. On the Stores page, click **Add New Store**. Then, select **GanosRaster(PG/PolarDB)** in the Raster Data Sources section.
3. Configure the connection information about the database.

   The following table lists the parameters.

Parameter	Description	Example
host	The IP address of the database instance or the endpoint of the RDS instance.	xxxxxxx.pg.rds.aliyuncs.com
port	The port used to connect to the database.	3432
database	The name of the database.	rasterdb
username	The username used to log on to the database.	pguser
password	The password used to log on to the database.	123456
schema	The schema of the table.	Default value: public.
table	The table name of the raster.	raster_table
column name	The column name of the raster.	raster_column
filter	The filter condition specified by the WHERE clause in the SQL statement to filter raster data. If multiple rasters meet the condition, the first one is used.	id=1 or name='srtm'
name	The name of the raster displayed in GeoServer.	myraster

REST API

- Create a data source.
  - Request URL: http://host:port/geoserver/rest/workspaces/{workspace}/coveragestores
  - Method: POST

- Parameters:
  - workspace: the name of the created workspace.
  - Request body: the information about the data store. Keep the default value of the type parameter, and set the url parameter to the connection information of the ApsaraDB RDS for PostgreSQL or ApsaraDB for PolarDB instance in JSON format. An example is as follows:

```
{
 "coverageStore": {
  "name": "<datasource_name>",
  "type": "GanosRaster(PG/PolarDB)",
  "enabled":"true",
  "workspace":"<wokrspace>",
  "url": "{\"column\":\"<raster_colunm>\",\"database\":\"<database_name>\",\"filter\":\"<raster_filter>\",\"host\":\"<pg_host>\",\"name\":\"<public_name>\",\"password\":\"<user_password>\",\"port\":<pg_port>,\"schema\":\"<schema_name>\",\"ssl\":false,\"table\":\"<raster_table_name>\",\"userName\":\"<user_name>\",\"valid\":true}"
 }
}
```

- Example:

```
{
 "coverageStore": {
  "name": "srtm",
  "type": "GanosRaster(PG/PolarDB)",
  "enabled":"true",
  "workspace":"test",
  "url": "{\"column\":\"rast\",\"database\":\"test_db\",\"filter\":\"name='srtm'\",\"host\":\"pgm-xxxxxxxx.pg.rds.aliyuncs.com\",\"name\":\"srtm_image\",\"password\":\"xxxx\",\"port\":3432,\"schema\":\"public\",\"ssl\":true,\"table\":\"raster_table\",\"userName\":\"raster_user\",\"valid\":true}"
 }
}
```

- Obtain the data source.
  - Request URL: http://host:port/geoserver/rest/workspaces/{workspace}/coveragestores
  - Method: GET

- Parameters:
  - workspace: the name of the created workspace.
  - Request body: the information about the data store. Keep the default value of the type parameter, and set the url parameter to the connection information of the ApsaraDB RDS for PostgreSQL or ApsaraDB for PolarDB instance in JSON format. An example is as follows:

```
{
 "dataStores": {
  "dataStore": [
   {
    "name": "txxxxx_shp_filter",
    "href": "http://11.xxx.xxx.xxx:8080/geoserver/rest/workspaces/tianxun/datastores/txxxxx_shp_filter.json"
   },
   {
    "name": "yxxxxx_shape1",
    "href": "http://11.xxx.xxx.xxx:8080/geoserver/rest/workspaces/tianxun/datastores/yxxxxx_shape1.json"
   },
   {
    "name": "yxxxxxn_shape2",
    "href": "http://11.xxx.xxx.xxx:8080/geoserver/rest/workspaces/tianxun/datastores/yxxxxxn_shape2.json"
   }
  ]
 }
}
```

- Publish a raster layer.
  - Request URL: http://host:port/geoserver/rest/workspaces/{workspace}/coveragestores/{store}/coverages
  - Method: POST
  - Parameter:
    - workspace: the name of the created workspace.
    - The body of the POST request is as follows:

```
{
 "coverage": {
  "abstract": "Digital elevation model for the Spearfish region.\r\n\r\nsfdem is a Tagged Image File Format with Geographic information",
  "defaultInterpolationMethod": "nearest neighbor",
  "description": "Generated from sfdem",
  "dimensions": {
   "coverageDimension": [
    {
     "description": "GridSampleDimension[-9.999999933815813E36,-9.999999933815813E36]",
     "name": "GRAY_INDEX",
     "range": {
      "max": -9.999999933815813e+36,
      "min": -9.999999933815813e+36
     }
    }
```

```
    ]
  },
  "enabled": true,
  "grid": {
   "@dimension": "2",
   "crs": "EPSG:26713",
   "range": {
    "high": "634 477",
    "low": "0 0"
   },
   "transform": {
    "scaleX": 30,
    "scaleY": -30,
    "shearX": 0,
    "shearY": 0,
    "translateX": 589995,
    "translateY": 4927995
   }
  },
  "interpolationMethods": {
   "string": [
    "nearest neighbor",
    "bilinear",
    "bicubic"
   ]
  },
  "keywords": {
   "string": [
    "WCS",
    "sfdem",
    "sfdem",
    "type\\@language=fr\\;\\@vocabulary=test\\;"
   ]
  },
  "latLonBoundingBox": {
   "crs": "EPSG:4326",
   "maxx": -103.62940739432703,
   "maxy": 44.5016011535299,
   "minx": -103.87108701853181,
   "miny": 44.370187074132616
  },
  "metadata": {
   "entry": [
    {
     "@key": "elevation",
     "dimensionInfo": {
      "enabled": false
     }
    },
    {
     "$": "10",
     "@key": "cacheAgeMax"
    },
    {
```

```
      "@key": "time",
      "dimensionInfo": {
       "defaultValue": "",
       "enabled": false
      }
     },
     {
      "$": "true",
      "@key": "cachingEnabled"
     },
     {
      "$": "sfdem_sfdem",
      "@key": "dirName"
     }
    ]
   },
   "name": "sfdem",
   "namespace": {
    "href": "http://localhost:8075/geoserver/restng/namespaces/sf.json",
    "name": "sf"
   },
   "nativeBoundingBox": {
    "crs": {
     "$": "EPSG:26713",
     "@class": "projected"
    },
    "maxx": 609000,
    "maxy": 4928010,
    "minx": 589980,
    "miny": 4913700
   },
   "nativeCRS": {
     "$": "PROJCS[\"NAD27 / UTM zone 13N\", \n  GEOGCS[\"NAD27\", \n   DATUM[\"North American Datum 1927\", \n    SPHEROID[\"Clarke 1866\", 6378206.4, 294.9786982138982, AUTHORITY[\"EPSG\",\"7008\"]], \n    TOWGS84[2.478, 149.752, 197.726, 0.526, -0.498, 0.501, 0.685], \n    AUTHORITY[\"EPSG\",\"6267\"]], \n   PRIMEM[\"Greenwich\", 0.0, AUTHORITY[\"EPSG\",\"8901\"]], \n   UNIT[\"degree\", 0.017453292519943295], \n   AXIS[\"Geodetic longitude\", EAST], \n   AXIS[\"Geodetic latitude\", NORTH], \n   AUTHORITY[\"EPSG\",\"4267\"]], \n  PROJECTION[\"Transverse_Mercator\", AUTHORITY[\"EPSG\",\"9807\"]], \n  PARAMETER[\"central_meridian\", -105.0], \n  PARAMETER[\"latitude_of_origin\", 0.0], \n  PARAMETER[\"scale_factor\", 0.9996], \n  PARAMETER[\"false_easting\", 500000.0], \n  PARAMETER[\"false_northing\", 0.0], \n  UNIT[\"m\", 1.0], \n  AXIS[\"Easting\", EAST], \n  AXIS[\"Northing\", NORTH], \n  AUTHORITY[\"EPSG\",\"26713\"]]",
     "@class": "projected"
   },
   "nativeFormat": "GeoTIFF",
   "nativeName": "sfdem",
   "requestSRS": {
    "string": [
     "EPSG:26713"
    ]
   },
   "responseSRS": {
    "string": [
     "EPSG:26713"
```

```
    ]
   },
   "srs": "EPSG:26713",
   "store": {
    "@class": "coverageStore",
    "href": "http://localhost:8075/geoserver/restng/workspaces/sf/coveragestores/sfdem.json",
    "name": "sf:sfdem"
   },
   "supportedFormats": {
    "string": [
     "ARCGRID",
     "IMAGEMOSAIC",
     "GTOPO30",
     "GEOTIFF",
     "GIF",
     "PNG",
     "JPEG",
     "TIFF"
    ]
   },
   "title": "Spearfish elevation"
  }
}
```

11.Desktop applications

11.1. Use OpenJump to access Ganos data

OpenJUMP is an open-source geographic information system (GIS) application written in Java. It provides the built-in map visualization feature for spatial analysis, such as buffer analysis, intersection computing, and union computing. In addition, its plug-in system allows you to customize the functionality as needed.

Add a connection to a Ganos database

1. Run OpenJUMP. In the top navigation bar, choose **File > Open**. In the dialog box that appears, click **Data Store Layer** in the left-side navigation pane.
2. In the **Connection Manager** dialog box that appears, click **Add**.
3. In the **Add Connection** dialog box that appears, set connection parameters.

   The following table describes the parameters.

Parameter	Description
Name	The custom name of the connection.
Driver	The driver of the connection. Set this parameter to **PostGIS**.
Server	The IP address of the database or the public network endpoint of the ApsaraDB for Relational Database Service (RDS) or PolarDB instance. You can obtain the endpoint from the Alibaba Cloud console.
Port	The port number of the database. You can obtain the port number from the Alibaba Cloud console.
Database	The name of the database to be connected.
User	The username used to log on to the database.
Password	The password used to log on to the database.

4. After setting the parameters, click **OK**. All layers in the connected Ganos database appear.

View and manage the geospatial data

After OpenJUMP is connected to the Ganos database, you can view and manage the geospatial data. Supported operations include zooming in, zooming out, walkthrough, and symbolic rendering.

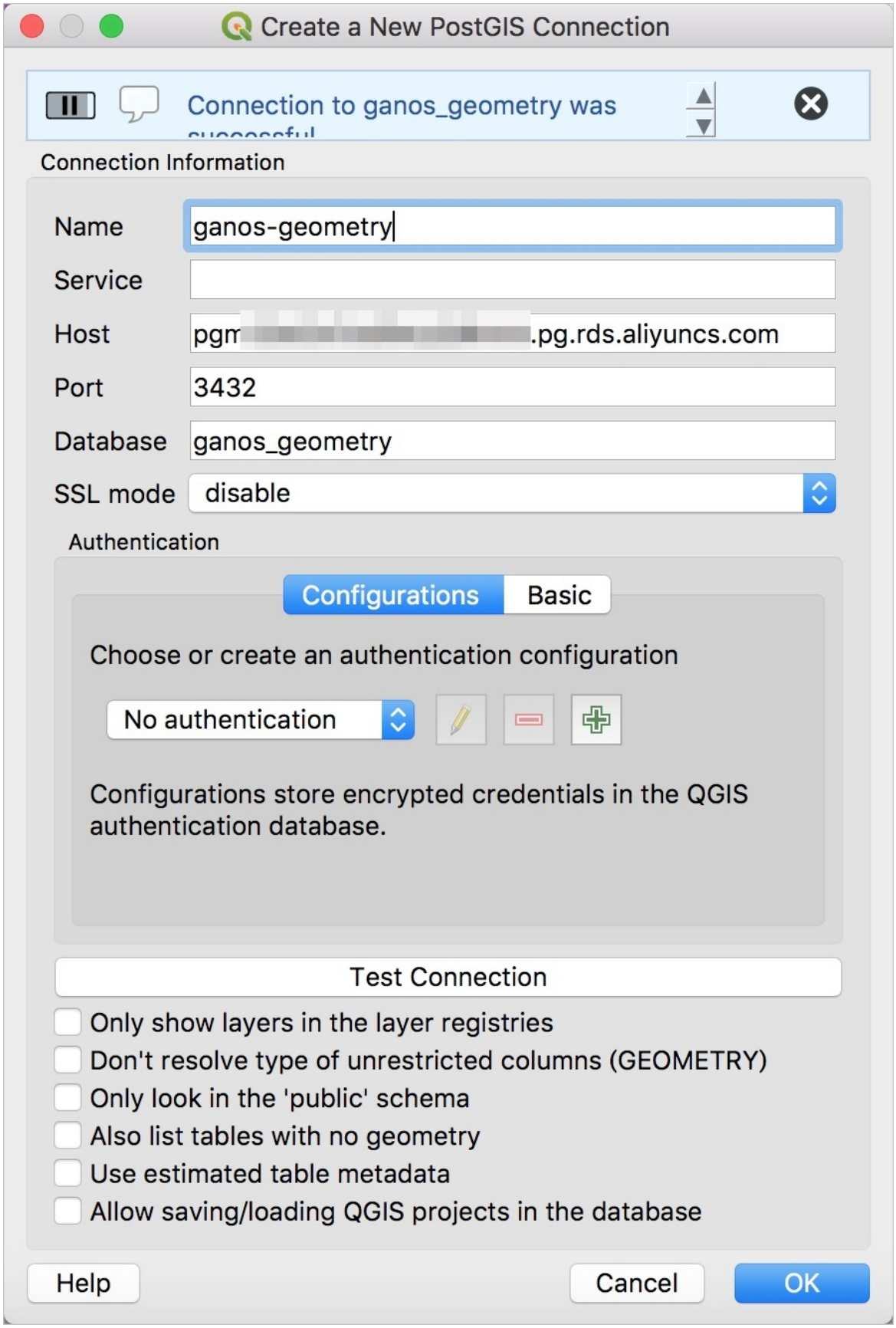
You can also query the attributes of a geometry object.

11.2. Use QGIS to access Ganos data

Quantum GIS (QGIS) is a user-friendly open-source desktop geographic information system (GIS) application. It supports multiple formats of raster images, vector images, and databases as data sources. Using QGIS, you can view, edit, manage, and analyze geospatial data, and create and export maps.

Add a connection to a Ganos database

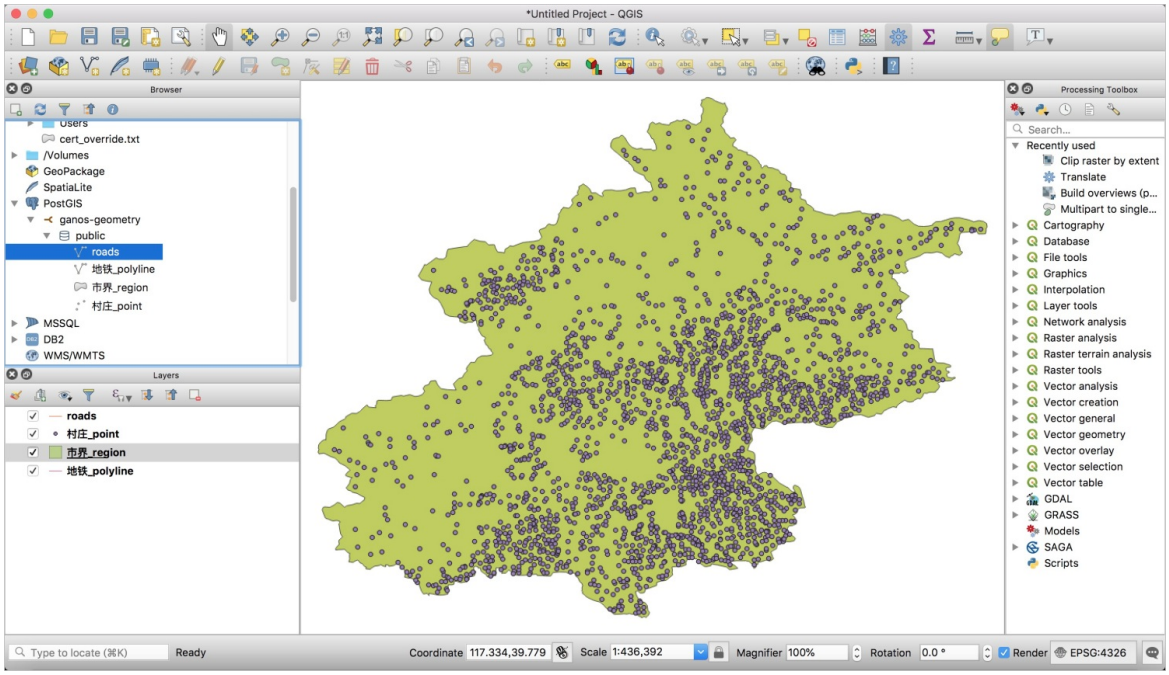
1. Run QGIS. In the Create a New PostGIS Connection dialog box, set connection parameters as required and click **Test Connection**.



The following table describes the parameters.

Parameter	Description
Name	The custom name of the connection.
Host	The IP address of the database or the public network endpoint of the ApsaraDB for Relational Database Service (RDS) or PolarDB instance. You can obtain the endpoint from the Alibaba Cloud console.
Port	The port number of the database. You can obtain the port number from the Alibaba Cloud console.
Database	The name of the database to be connected.
SSL mode	The Secure Sockets Layer (SSL) encryption mode. Set this parameter to **disable**.

2. Select a connected Ganos database. You can view the existing database and layers under the database. Double-click a layer to browse, view, and manage the geospatial data in the database.

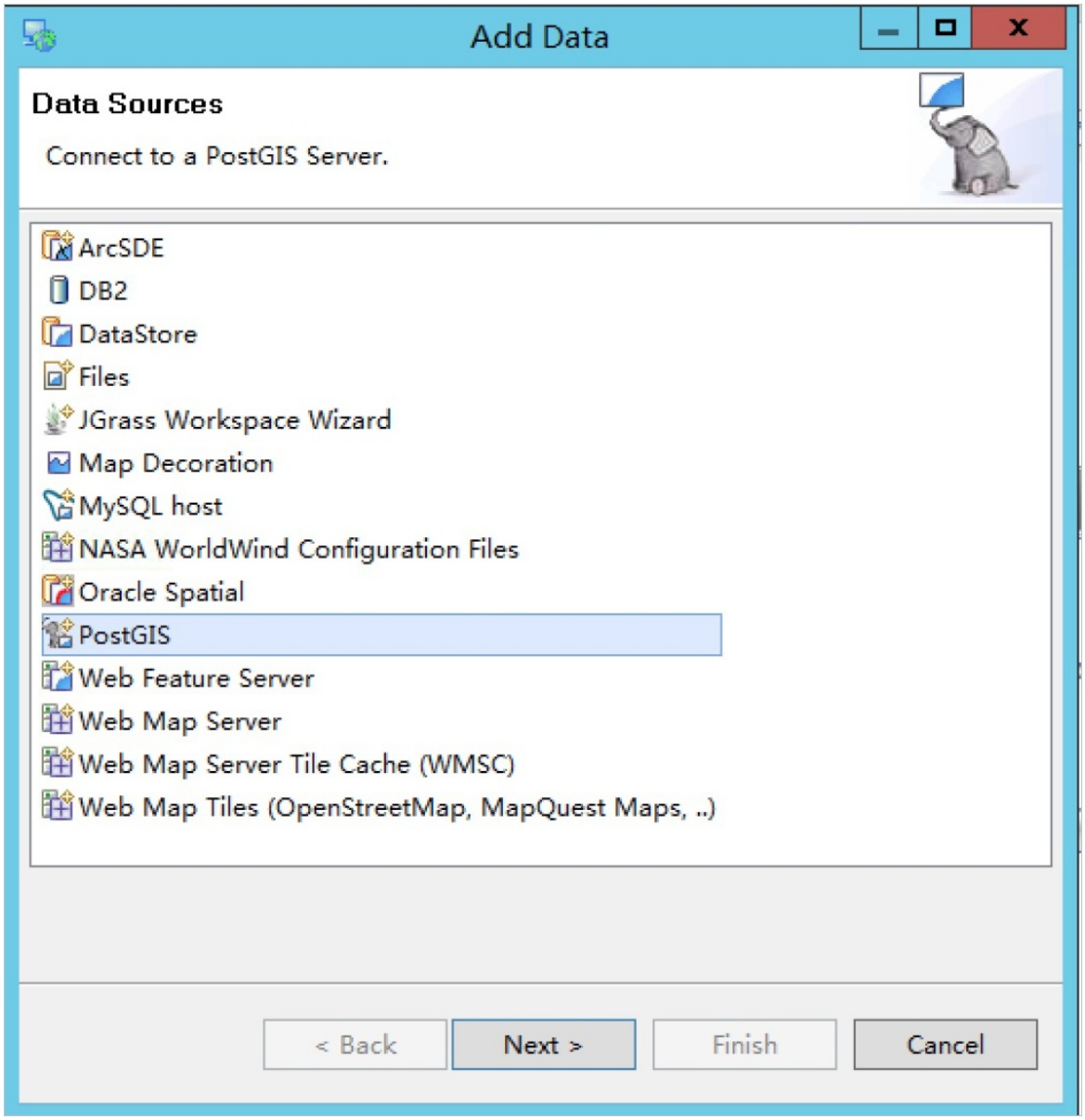


11.3. Use uDig to access Ganos data

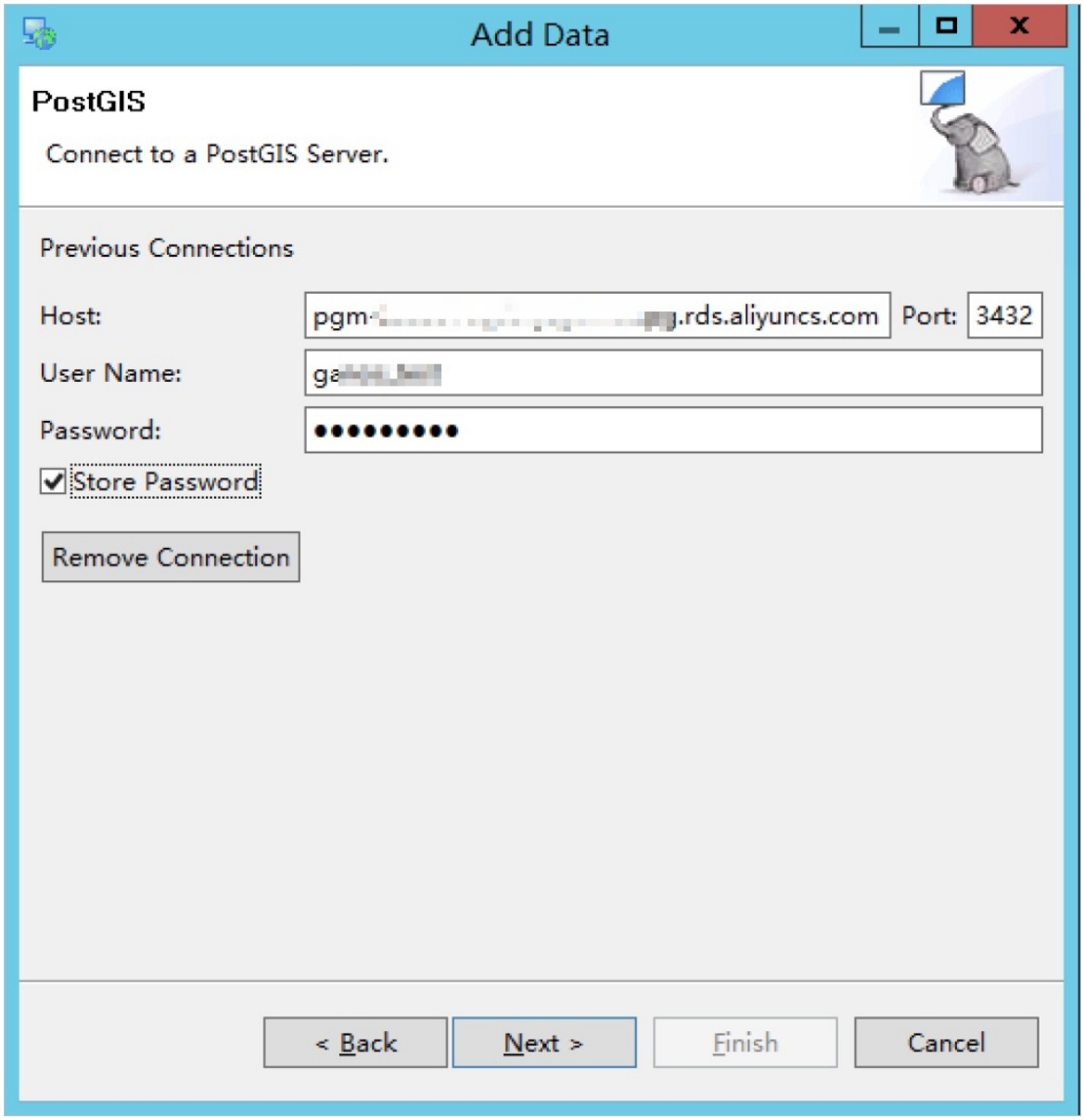
uDig is an open-source desktop geographic information system (GIS) application for editing and viewing geospatial data, such as a map file in the shapefile format. It supports OpenGIS standards and provides advanced Internet GIS, Web Map Service (WMS), and web servers. Based on Java, uDig allows you to use open-source plug-ins to view and manage geospatial data.

Add a connection to a Ganos database

1. Run uDig. In the top navigation bar, choose **Layer > Add Data**. In the dialog box that appears, select PostGIS and click Next.



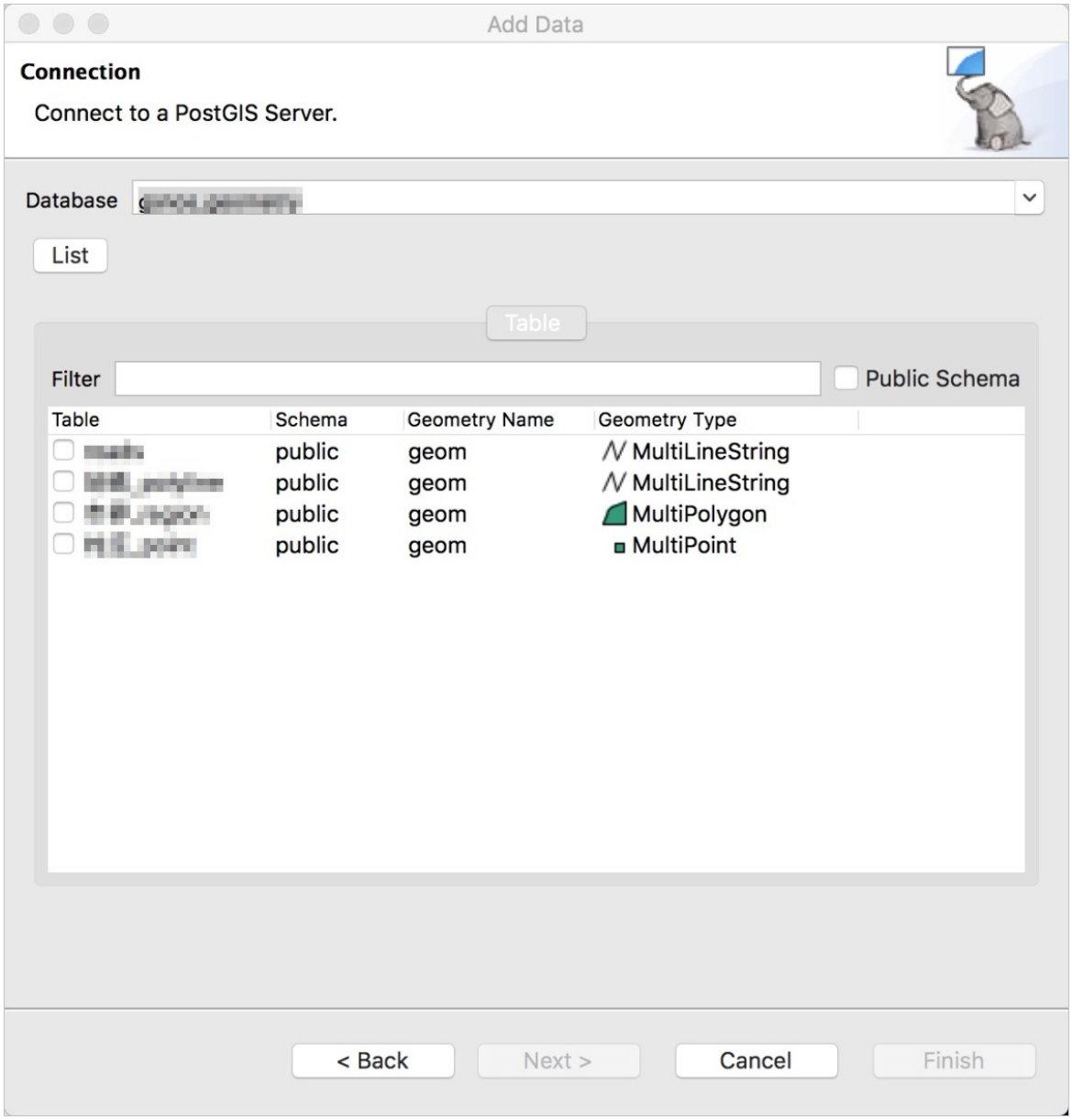
2. Set the connection information about the database.



The following table describes the parameters.

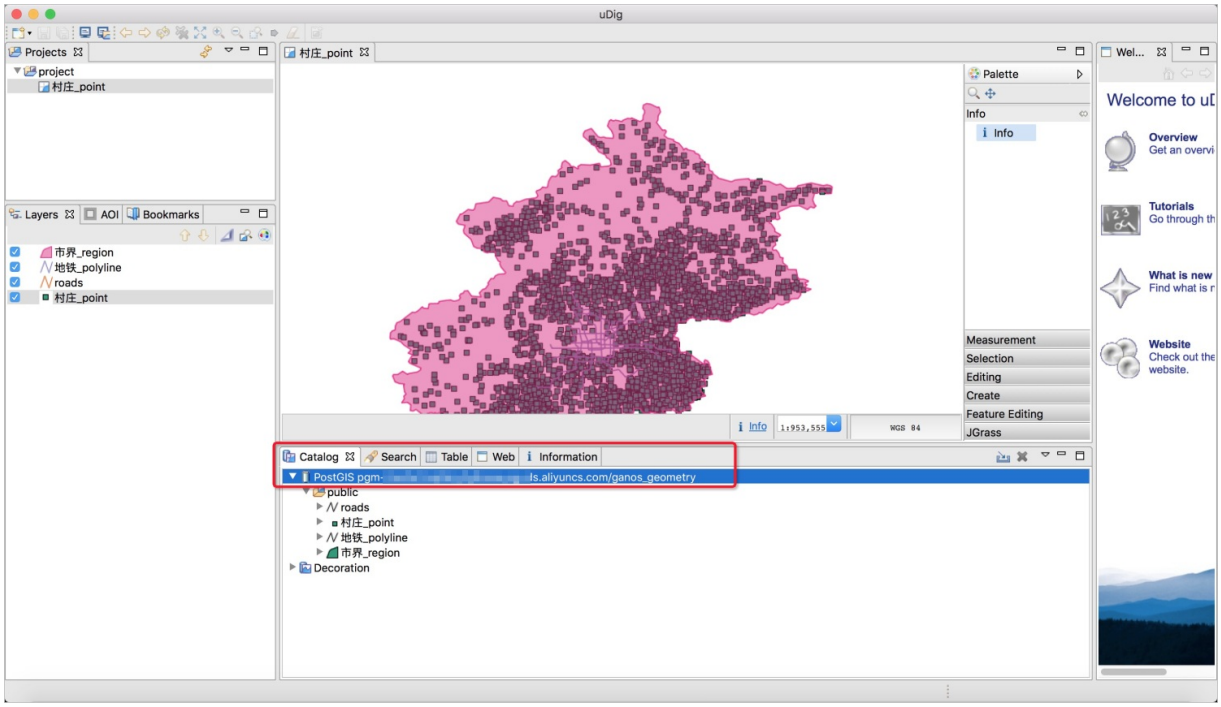
Parameter	Description
Host	The IP address of the database or the public network endpoint of the ApsaraDB for Relational Database Service (RDS) or PolarDB instance. You can obtain the endpoint from the Alibaba Cloud console.
Port	The port number of the database. You can obtain the port number from the Alibaba Cloud console.
User Name	The username used to log on to the database.
Password	The password used to log on to the database.

3. Select a database from the Database drop-down list.



View and manage the geospatial data

After uDig is connected to the Ganos database, you can browse, view, and manage the geospatial data in the Ganos database.


uDig
Projects
project
村庄_point
Layers
AOI
Bookmarks
市界_region
地铁_polyline
roads
村庄_point
Palette
Info
Measurement
Selection
Editing
Create
Feature Editing
JGrass
Catalog
Search
Table
Web
Information
public
roads
村庄_point
地铁_polyline
市界_region
Decoration
Welcome to uD
Overview
Tutorials
What is new
Website

12.FAQ

12.1. Load raster data

This topic describes how to load a raster data file from your computer or an Alibaba Cloud Object Storage Service (OSS) to Ganos by using the ST_ImportFrom function.

Precautions

Your RDS instance is authorized to access the raster data file. If the raster data file is stored in an OSS bucket, your RDS instance resides in the same region as the OSS bucket. For more information, see OSS domain names.

Introduction to the ST_ImportFrom function

The ST_ImportFrom function is used to load a raster data file from your computer or an OSS bucket to Ganos.

> Note
> - In addition to the ST_ImportFrom function, Ganos provides other raster-related functions. For more information, see Raster SQL Reference.
> - For more information about the parameters in the ST_ImportFrom function, see ST_ImportFrom.

Syntax

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

Examples

1. Create the ganos_raster extension.

   ```
   CREATE EXTENSION Ganos_Raster CASCADE
   ```

2. Load a raster data file from your computer to Ganos.

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

3. Load a raster data file from an OSS bucket to Ganos.

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

12.2. Load vector data

This topic describes how to load a vector data file into Ganos by using shp2pgsql, ogr2ogr, or QGIS.

Preparations

Before you load a vector data file, run the following command on your AnalyticDB for PostgreSQL instance to create the ganos_geometry extension:

```
CREATE EXTENSION ganos_geometry CASCADE
```

Load vector data by using shp2pgsql

The shp2pgsql command line tool is used to convert Esri shapefiles into SQL files. For more information, visit shp2pgsql. Then, you can load the SQL files into Ganos.

- Syntax

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

- Parameters

Parameter	Description	Example
-s <srid>	The spatial reference system identifier (SRID) of the vector data file.	4490
-c	Specifies to create a table. You can also use the following parameters: ○ -a: appends data to a table. ○ -d: deletes a table before data is loaded. ○ -p: creates a schema in SQL without writing spatial data.	None
-W <charset>	The character set that is used by the shapefile.	GBK or UTF8
<path_to_shpfile>	The storage path of the shapefile. The storage path must be accessible to the shp2pgsql command line tool.	/home/roads.shp
<schema>. <table_name>	The name of the geometry table that you created.	public.roads
-d <dbname>	The name of the database.	mydb
-h <host>	The endpoint of the instance on which to create the database.	pgm-xxxxxxx.pg.rds.aliyuncs.com
-U <user_name>	The username that is used to connect to the database.	my_name
-p <port>	The port number that is used to connect to the database.	3433

- Example

```
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
```

> **Note** If you want to import more than one vector data file at a time by using a script, run the `export PGPASSWORD=my_pass` command to set the PGPASSWORD environment variable to my_pass.

Load vector data by using ogr2ogr

The ogr2ogr command line tool is provided by GDAL/OGR to convert data. It supports common vector data types, such as Esri ShapFile, MapInfo, and FileGDB. For more information about ogr2ogr, visit ogr2ogr. For more information anout vector data types, visit Vector drivers.

> **Note** Before you load vector data by using the ogr2ogr command line tool, make sure that GDAL supports the PostGIS vector driver.

- Syntax

```
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
```

- Parameters

Parameter	Description	Example
-nln <table_name>	The name of the vector data file.	roads
-f PostgreSQL PG:"dbname='<dbname>' host='<host>' port='<port>' user='<user_name>' password='<password>'"	The information that is used to connect to the AnalyticDB for PostgreSQL instance.	dbname='mydb' host='pgm-xxxxxxx.pg.rds.aliyuncs.com' port='3433' user='my_name' password='my_pass'
-lco SPATIAL_INDEX=GIST	The GiST spatial index that you want to create.	None
-lco FID=<FID_NAME>	The ID of the specified feature.	fid
-overwrite	The rewrite mode that you want to use. You can also select the -append mode.	None
<PATH_TO_FILE>	The storage path of the vector data file. Make sure that the storage path is accessible to the ogr2ogr command line tool.	/home/roads.shp /home/land.tab

Parameter	Description	Example
-nlt GEOMETRY	Specifies the geometry data type. If the vector data file contains polygon data, we recommend that you specify this parameter. This allows you to avoid errors that occur if the vector data file contains not only data of the MultiPolygon type but also data of other types.	None

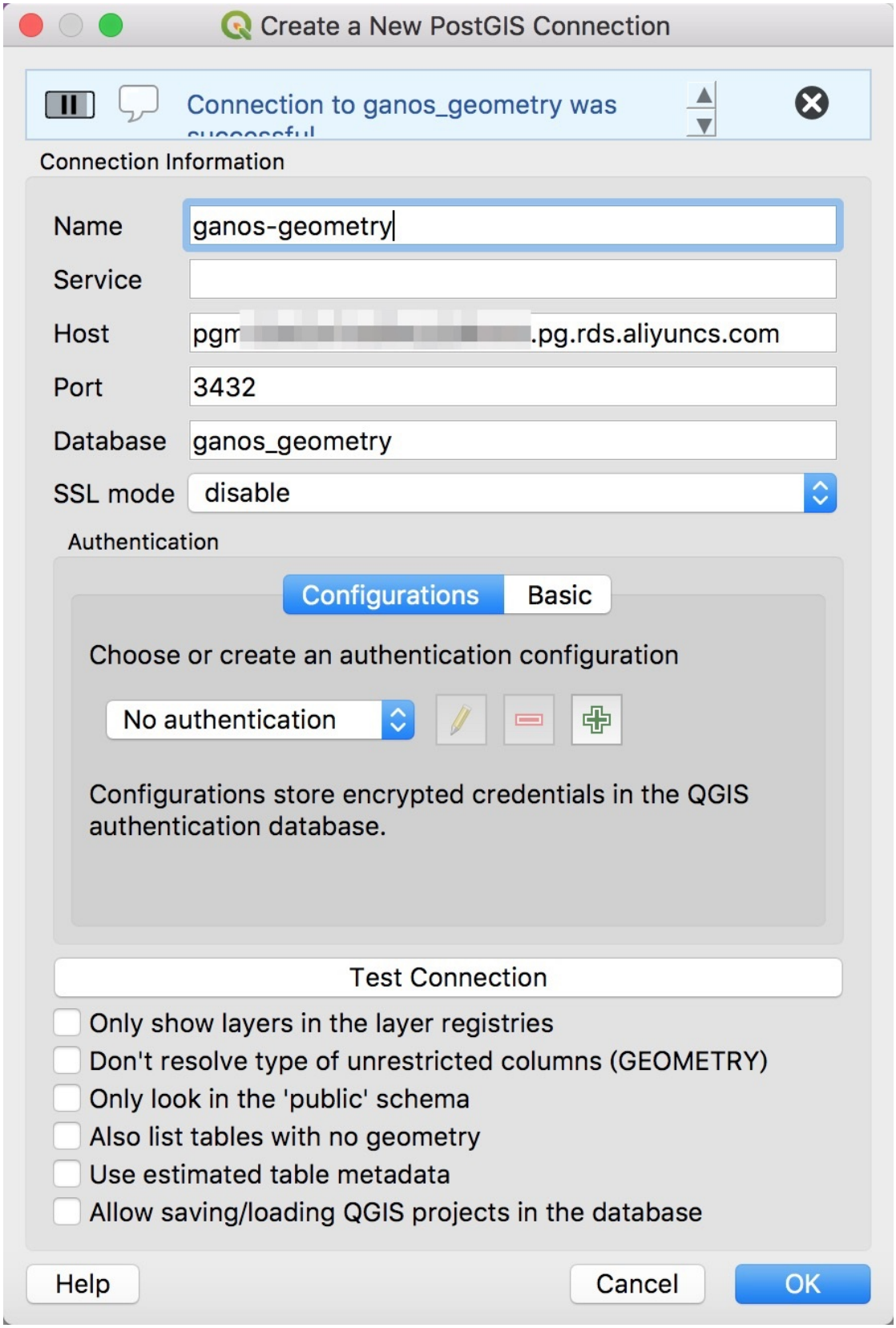
- Example

```
ogr2ogr -nln roads -f PostgreSQL PG:"dbname='mydb' host='pgm-xxxxxxx.pg.rds.aliyuncs.com' port='343
3' user='my_name' password='my_pass'" -lco SPATIAL_INDEX=GIST -lco FID=fid -overwrite "/home/roads.
shp" -nlt GEOMETRY
```

Load vector data by using QGIS

QGIS was known as Quantum GIS. It is a user-friendly, open-source desktop tool that supports various vector and raster formats. It allows you to use databases as data sources. In addition, it allows you to visualize, manage, and analyze data and to prepare maps. For more information, see QGIS.

1. Establish a connection to Ganos.
   i. In the right-side **Browser** pane, create a PostGIS connection.
   ii. Configure the required parameters.



The following table describes the parameters.

Parameter	Description
Name	The name of the connection.
Host	The IP address of your ECS instance or the public endpoint of your ApsaraDB RDS instance or PolarDB cluster. You can obtain this information from the Alibaba Cloud Management Console.
Port	The port number of your ECS instance, ApsaraDB RDS instance, or PolarDB cluster. You can obtain this information from the Alibaba Cloud Management Console.
Database	The name of the database.
SSL Mode	Specifies whether to establish an encrypted connection based on SSL.

iii. Click **Test Connection**. Then, wait until the connection is established.

2. Load the vector data file.
   i. In the menu bar, choose **Database > DB Manager**.
   ii. In the dialog box that appears, select the data source from which you want to import the vector data file, and click **Import Layer/File...**.
   iii. In the dialog box that appears, click the vector data file, configure the required parameters, and then click **OK**.

12.3. Trajectory FAQ

This topic provides answers to the commonly asked questions about trajecotry. This helps you use the Ganos engine in AnalyticDB for PostgreSQL.

How do I convert the data of coordinate points into a trajectory object?

You can use the ST_makeTrajectory constructor to convert the data of coordinate points into a trajectory object. You can execute the following statements:

```
-- Create an extension.
create extension ganos_trajectory cascade;
-- Create a point table.
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);
-- Create a trajectory table.
create table traj(id integer, traj trajectory);
-- Insert data.
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;
```

What shall I do if the built-in ST_makeTrajectory constructor does not meet my requirements?

If the built-in ST_makeTrajectory constructor does not meet your requirements, you can customize one with required attributes. For example, to create a trajectory object with five attributes, including two of the int8 type, two of the float4 type, and one of the timestamp type, you can customize the following constructor:

```
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;
```

In this constructor, the first six parameters are fixed, and the last five parameters are customized attributes. You can directly use this constructor with the customized attributes as a user-defined overloaded function.

How do I append trajectory points to a trajectory object?

You can use the ST_append constructor to append trajectory points to an existing trajectory object. The following statements describe the syntax of the ST_append constructor.

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

The following example shows how to append trajectory points to a trajectory object based on a point table.

```
-- Create an extension.
create extension ganos_trajectory cascade;
```

```
-- Create a point table.
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);
-- Create a trajectory table.
create table traj(id integer, traj trajectory);
-- Insert data.
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;
-- Insert trajectory points.
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);
-- Update the trajectory object based on a single trajectory point.
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;
-- Use a generated SQL script to update the trajectory object.
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;
-- Update the trajectory object based on multiple trajectory points.
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;
```

How do I enable advanced compression?

LZ4 is an advanced compression algorithm, which has a higher compression ratio and execution speed. To enable and disable the LZ4 compression algorithm, execute the following statements:

```
-- Enable LZ4 compression.
Set toast_compression_use_lz4 = true;
-- Disable LZ4 compression and use the default PostgreSQL compression algorithm.
Set toast_compression_use_lz4 = false;
```

To enable and disable the LZ4 compression algorithm for the entire database by default, execute the following statements:

```
-- Enable LZ4 compression for the database.
Alter database dbname Set toast_compression_use_lz4 = true;
-- Disable LZ4 compression and use the default compression algorithm for the database.
Alter database dbname Set toast_compression_use_lz4 = false;
```

How do I set a default length for string-type attribute fields?

You can use the GUC variable ganos.trajectory.attr_string_length to set a default length for string-type attribute fields. You can execute the following statement:

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

How do I calculate the maximum, minimum, or average value of an attribute field?

To calculate the maximum, minimum, or average value of an attribute field, execute the following statement:

```
With traj AS (
 select '{"trajectory":{"version":1,"type":"STPOINT","leafcount":2,"start_time":"2010-01-01 11:30:00","end
_time":"2010-01-01 12:30:00","spatial":"SRID=4326;LINESTRING(1 1,3 5)","timeline":["2010-01-01 11:30:00","
2010-01-01 12:30:00"],"attributes":{"leafcount":2,"velocity":{"type":"integer","length":4,"nullable":true,"va
lue":[1,100]},"speed":{"type":"float","length":8,"nullable":true,"value":[null,1.0]},"angel":{"type":"string","l
ength":64,"nullable":true,"value":["test",null]}, "tngel2":{"type":"timestamp","length":8,"nullable":true,"v
alue":["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;
```

13.Best practice

13.1. Trajectory best practices

Use appropriate spatio-temporal indexes

You can create appropriate indexes based on application requirements to accelerate queries. Trajectory data supports the following index types:

- Spatial index: the index on space, which applies when you query only the spatial data of a trajectory.
- Temporal index: the index on time, which applies when you query only the time range of a trajectory.
- Spatio-temporal composite index: the composite index on both space and time, which applies when you query both the spatial data and time range of a trajectory.

```
-- Create a function-based spatial index to accelerate the filtering of spatial data.
create index tr_spatial_geometry_index on trajtab using gist (st_trajectoryspatial(traj));
-- Create a function-based temporal index to accelerate the filtering of time.
create index tr_timespan_time_index on trajtab using gist (st_timespan(traj));
-- Create function-based indexes on the start time and end time of a trajectory object.
create index tr_starttime_index on trajtab using btree (st_starttime(traj));
create index tr_endtime_index on trajtab using btree (st_endtime(traj));
-- Create a btree_gist extension.
create extension btree_gist;
-- Use btree_gist to create a spatio-temporal composite index on the start time, end time, and spatial data of a trajectory object.
create index tr_traj_test_stm_etm_sp_index on traj_test using gist (st_starttime(traj),st_endtime(traj),st_trajectoryspatial(traj));
```

Use appropriate partitioned tables

The amount of trajectory data in a database keeps increasing along with the continuous use of the database. As a result, a larger number of database indexes are created and data queries slow down. In this case, you can use partitioned tables to decrease the data size of a single table.

For more information about how to use partitioned tables, see Table partitioning.

Reduce the use of string-type attribute fields

A large number of string-type attribute fields in trajectory data lead to a waste of the storage space and performance deterioration.

- If string-type attribute fields have fixed values, they can be converted into integers. We recommend that you convert the data type in code.
- If string-type attribute fields are required, you can set a default length for the fields to save space.

  To set a default length for string-type attribute fields, execute the following statement:

  ```
  -- Set the default length of string-type attribute fields to 32.
  Set ganos.trajectory.attr_string_length = 32;
  ```

Use multiple trajectory points to generate a trajectory object

We recommend that you use multiple trajectory points to generate a trajectory object and avoid appending trajectory points one by one.

Use the advanced compression mode

LZ4 is an advanced compression algorithm, with a higher compression ratio and execution speed. To enable and disable the LZ4 compression algorithm, execute the following statements:

```
-- Enable LZ4 compression.
Set toast_compression_use_lz4 = true;
-- Disable LZ4 compression and use the default PostgreSQL compression algorithm.
Set toast_compression_use_lz4 = false;
```

To enable and disable the LZ4 compression algorithm for the entire database by default, execute the following statements:

```
-- Enable LZ4 compression for the database.
Alter database dbname Set toast_compression_use_lz4 = true;
-- Disable LZ4 compression and use the default compression algorithm for the database.
Alter database dbname Set toast_compression_use_lz4 = false;
```

13.2. Use Ganos to create spatial indexes in parallel

PolarDB for Oracle allows you to use Ganos to create spatial indexes in parallel. Ganos uses the spatial sorting (GiST Sort) method to create indexes in parallel and reduce the number of read and write operations on disks. This way, the indexing efficiency is improved. This topic provides an example to show how you can accelerate the indexing process for millions of spatio-temporal data records.

Context

PostgreSQL databases are suitable for storing and managing spatial data. An increase in data volume can increase performance issues. It is time-consuming to create spatial indexes for tens of millions of data records.

Prerequisites

- An Alibaba Cloud account is created.
- A cluster of PolarDB for Oracle is created. In the example described in this topic, a cluster of PolarDB for Oracle that has 4 cores and 16 GB memory is used.

Additional considerations

The GiST Sort method is suitable only for point data. If you use this method for other types of spatial data, the query performance of indexes can be compromised.

Procedure

1. Create a spatio-temporal extension and generate test data.
   i. Create a database in the PolarDB for Oracle cluster.
   ii. Use pgAdmin to connect to the database.

iii. Execute the following statement to create a Ganos spatio-temporal extension:

```
create extension ganos_geometry cascade;
```

iv. Execute the following statements to generate test data:

```
CREATE TABLE test (id int, geom Geometry(Point, 4326), name text, code int);
INSERT INTO test SELECT i,
        ST_SetSRID(ST_MakePoint(random() * 180.0, random() * 90.0), 4326),
        'text', 1
        FROM generate_series(1,1000 * 10000) AS i;
```

2. Use the traditional method to create a spatial index.

   Execute the following statements to create a spatial index:

```
select now();
CREATE index ON test using GiST(geom);
select now();
```

   The following number shows the amount of time used to create the spatial index:

```
330.10 S
```

3. Use Ganos to create a spatial index in parallel.

   Execute the following statements to create a spatial index in parallel by using Ganos:

```
set max_parallel_maintenance_workers=4;
set maintenance_work_mem='1GB';
set polar_enable_gist_sort= on;
select now();
CREATE index on test using GiST(geom);
select now();
```

   The following number shows the amount of time used to create the spatial index:

```
33.54 S
```

Conclusion

Compared with the traditional method, the GiST Sort method improves the indexing efficiency for spatial data by approximately nine times.