ALIBABA CLOUD

阿里云

云数据库RDS

时空数据库

文档版本：20211103

阿里云

法律声明

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

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。

2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。

3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。

4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的“现状”、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。

5. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含“阿里云”、“Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。

6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

格式	说明	样例
危险	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	危险 重置操作将丢失用户配置数据。
警告	该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。	警告 重启操作将导致业务中断，恢复业务时间约十分钟。
注意	用于警示信息、补充说明等，是用户必须了解的内容。	注意 权重设置为0，该服务器不会再接受新请求。
说明	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	说明 您也可以通过按Ctrl+A选中全部文件。
>	多级菜单递进。	单击**设置**> **网络**> **设置网络类型**。
粗体	表示按键、菜单、页面名称等UI元素。	在**结果确认**页面，单击**确定**。
Courier字体	命令或代码。	执行 `cd /d C:/window` 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	`bae log list --instanceid` *Instance_ID*
[] 或者 [a|b]	表示可选项，至多选择一个。	`ipconfig [-all|-t]`
{} 或者 {a|b}	表示必选项，至多选择一个。	`switch {active|stand}`

目录

1.简介

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

概述

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

本文档介绍如何使用Ganos对时空数据进行管理和分析。

如需人工帮助，可以在RDS管理控制台的右上角选择**工单 > 提交工单**。

如果业务复杂，您也可以购买支持计划，获取由IM企业群、技术服务经理（TAM）、服务经理等提供的专属支持。

定价

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

钉钉官方群

您可以使用手机钉钉扫描下方二维码加入官方群，获取更多资讯。

2.时空数据库 Release Notes

本文介绍时空数据库（Ganos）的版本更新说明。

小版本	说明

4.1	• 性能优化
	  ○ 增强ST_ImportFrom函数，增加OSS文件映射为内存文件功能，提升数据导入速度。
	  ○ GisT索引和TrajGisT索引新增算子族trajgist_ops_multi，用于多外包框索引键值。
	  ○ 新增矢量金字塔支持元数据信息查询。
	• Bug修复
	  ○ 修复轨迹对象执行相交判断时内存泄露问题。

4.0	• 性能优化
	  ○ 增强栅格表面计算函数（ST_Hillshade、ST_Slope和ST_Aspect）返回Raster对象取代像素矩阵。
	  ○ 增强ST_Split按照指定的规则对轨迹对象进行切分返回子轨迹数组。
	• Bug修复
	  修复ST_SrEqual在某些条件下判断失败的问题。

3.9	• 新特性
	  ○ 新增ST_removeDriftPoints函数，使用指定规则删除轨迹中的漂移点。
	  ○ 新增ST_Split函数，使用指定的几何对象对轨迹进行切分。
	  ○ 新增ST_ExportTo函数，将轨迹导出到外部文件存储，数据库内仅保留元数据。
	  ○ 新增ST_IsExternal函数，检查轨迹是否存储于外部文件。
	  ○ 新增ST_importFrom函数，将外部存储模式的轨迹重新转化为存储在数据库内的轨迹。
	  ○ 新增ST_StorageLocation函数，返回轨迹存储在外部的位置。
	  ○ 新增ST_AKID函数，返回读取轨迹时，与OSS交互时所使用的AccessKeyID。
	  ○ 新增ST_SetAccessKey函数，设置读取轨迹时，与OSS交互时所使用的AccessKey（包括AccessKeyID和AccessKeySecret，类似用户名与密码）。
	  ○ 新增ST_SetAkId函数，设置读取轨迹时，与OSS交互时所使用的AccessKeyID。
	  ○ 新增ST_SetAkSecret函数，设置读取轨迹时，与OSS交互时所使用的AccessKeySecret。
	  ○ 新增ST_SetStorageLocation函数，设置读取轨迹时，外部文件的存储位置。
	  ○ 新增ST_DeleteGTF函数，删除指定文件夹下所有的轨迹导出的文件。
	• Bug修复
	  ○ 修复创建矢量金字塔使用字符串方式过滤失败问题。
	  ○ 修复QGIS连接时无法列出图层列表的问题。

小版本	说明

3.8	• 新特性
	  ○ 新增ST_AsDatasetFile函数，支持将指定范围的栅格对象以指定文件格式的二进制流进行返回。
	  ○ 新增ST_RasterDrivers函数，支持对所有栅格数据驱动的状态进行查询。
	  ○ 新增Ganos_FDW模块，支持通过fdw方式访问存储于OSS上的矢量空间数据。
	  ■ 新增ST_FDWDrivers函数获得所有Ganos FDW支持的数据源驱动列表。
	  ■ 新增ST_ForeignTables函数查询外部数据源中表的名称。
	  ■ 新增ST_RegForeignTables函数将数据源中的表注册为外表。
	• Bug修复
	  修复ST_Clip函数指定空间参考进行重投影操作失败的问题。

3.7	• 性能优化
	  ST_SubRaster函数支持栅格像元类型转换以及像元值拉伸。支持多波段遥感影像转三波段图像的AI识别场景。
	• Bug修复
	  ○ 修复栅格解析无效的直方图信息可能会导致数据库崩溃的问题。
	  ○ 修复进行投影变化操作后，数据库退出时可能会崩溃的问题。
	  ○ 修复栅格数据导入时可能会出现崩溃的问题。

3.6	• 新特性
	  ○ 新增ST_RPCGeoreference函数，用于获取栅格数据RPC（Rational Polynomial Coefficients）信息。
	  ○ 新增ST_SetRPCGeoreference函数，用于设置栅格数据RPC信息。
	  ○ 新增ST_RPCRectify函数，用于根据栅格影像的RPC参数对栅格进行校正操作，返回校正后的栅格对象。
	  ○ 新增使用并行方式创建GisT索引。
	• Bug修复
	  修复使用pg_dump时自定义空间参考可能无法导出的问题。

3.5	• 性能优化
	  使用栅格对象对一个具有分块数据的栅格对象进行更新，系统会自动删除原有栅格对象的分块数据。
	• Bug修复
	  ○ 修复Trajectory扩展无法升级的问题。
	  ○ 修复某些情况下，栅格对象采用Average重采样时出错的问题。
	  ○ 修复轨迹对象中如果多个时间戳的轨迹点相同，查询结果可能不正确的问题。
	  ○ 修复Geos对象转换失败后，直接退出的问题，同时对转换失败的原因进行提示。

小版本	说明

3.4	• 新特性
	  ◦ 为了加速时空数据索引构建、提高空间查询效率，优化时空对象的存储模式，支持使用时空大对象特征签名，新增polar_enable_storage_partial参数，支持时空对象行内与行外组合存储。具体内容，请参见时空大对象特征签名。
	  ◦ 新增ST_ImportFrom函数，支持栅格数据并行化导入。
	  ◦ 新增ST_BuildPyramid函数，支持栅格数据并行创建金字塔。
	  ◦ 新增Trajectory Empty对象，支持某些场景下不符合要求而返回NULL对象。
	• Bug修复
	  ◦ 修复ST_AddZ函数在16BSI情况下可能结果不正确的问题。
	  ◦ 修复Trajectory模块在某些情况下无法升级的问题。
	  ◦ 修复Trajectory模块中部分函数无法利用索引的问题。
	  ◦ 修复部分nd函数没有正确处理时间段不相交的场景，导致返回error而非false的问题。

3.3	• 新特性
	  ◦ 新增ST_JaccardSimilarity函数，用于计算轨迹对象相似度。
	  ◦ 新增ST_Transform函数，用于将轨迹从一个空间参考系转换到另一个空间参考系。
	• 性能优化
	  创建矢量金字塔时，支持用户指定创建范围，解决在部分场景下无法自动计算空间范围的问题。
	• Bug修复
	  修复某些环境下，Ganos数据目录设置不正确的问题。

小版本	说明

3.2	• 新特性
	  ○ 新增矢量金字塔返回图片格式（基于流形式）功能，用于矢量数据的快速图形化显示。
	  ○ 新增栅格数据类型JPEG2000压缩算法，支持16bit栅格数据压缩存储。
	  ○ 新增ganos_update函数，用 `select ganos_update();` 命令可以升级所有的Ganos插件到最新版本。
	  ○ 新增Trajectory数据类型：
	  ■ 支持原生时空索引。
	  ■ 新增Gist索引支持索引轨迹类型，并提供六种不同维度的算子族以支持不同维度的分析需求。
	  ■ 新增时空外包框类型BoxND，可用于时空范围表示以及存储轨迹。
	  ■ 新增对应不同维度的相交（&&）、包含（@>）、被包含（<@）算子。
	  ■ 新增ST_ndIntersects、ST_ndDWithin、ST_ndContains、ST_ndWithin四类轨迹处理函数。
	  ■ 对轨迹类型提供统计信息收集功能，以及根据统计信息预估扫描代价功能。
	  ■ 提供新的索引方式TrajGist，提供更好的索引选择。
	• 性能优化
	  ○ 优化st_dwithin距离查询，提升查询性能。
	  ○ 优化时空范围查询，GIST索引二阶段查询优化，提升查询性能。
	  ○ 矢量金字塔功能改进：
	  ■ 支持任意srid坐标的源数据，支持3857和4326两种瓦片输出。
	  ■ 新增pixelSize参数设置，对点数据进行聚合，减少瓦片的数量。
	• Bug修复
	  ○ 修复轨迹数据类型时间相交错误问题。
	  ○ 修复某些情况下更新Ganos Raster失败问题。
	  ○ 修复Ganos二进制文件更新到新版本后可能出现崩溃的问题。
	  ○ 修复用默认参数构建矢量金字塔点数据后，顶级瓦片数据量过大的问题。

	• 新特性
	  ○ 新增支持具有SubSet的NetCDF数据类型数据，可按照指定的图层名称导入。
	  ○ 新增支持栅格数据自定义元数据以及时序信息：
	  ■ 新增ST_MetaItems函数，用于获取所有的自定义元数据项目名称
	  ■ 修改ST_MetaData函数， 用于获取自定义元数据项以及返回以JSON方式表达的元数据项。
	  ■ 新增ST_SetMetaData函数，用于设置元数据项。
	  ■ 新增ST_BeginDateTime函数，用于获取栅格数据的起始时间。
	  ■ 新增ST_EndDateTime函数，用于获取栅格数据的终止时间。
	  ■ 新增ST_SetBeginDateTime函数，用于设置栅格数据的开始时间。
	  ■ 新增ST_SetEndDateTime函数，用于设置栅格数据的结束时间。
	  ■ 新增ST_SetDateTime函数，用于设置栅格数据的开始、结束时间以及波段获取时间。

小版本	说明
3.0	○ 新增支持栅格数据返回基于流形式的图片格式： ■ 新增ST_AsImage函数，用于获取基于流形式的图片格式。 ■ 新增ST_AsPNG函数，用于获取基于流形式的PNG图片格式。 ■ 新增ST_AsJPEG函数，用于获取基于流形式的JPEG图片格式。 ○ 新增支持空间网格数据类型以及操作运算： ■ 新增geomgrid数据类型。 ■ 新增ST_AsText函数，用于将网格数据类型转换为文本表示方式。 ■ 新增ST_AsGeometry函数，用于将网格数据类型转换为几何数据类型。 ■ 新增ST_AsBinary函数，用于将网格数据类型转换为二进制数据类型。 ■ 新增ST_AsBox函数，用于将网格数据量类型转换为BOX数据类型。 ■ 新增ST_AsGrid函数，用于计算几何数据类型所对应的几何网格数据。 ■ 新增ST_GridFromText函数， 用于将基于文本表示网格转换为几何网格数据类型。 ■ 新增ST_GridFromBinary函数，用于将基于二进制的表示的网格转换为几何网格数据类型。 ■ 新增ST_Intersects函数，用于判断网格数据类型与几何数据类型是否相交。 ■ 新增ST_Contains函数，用于判断网格数据与网格数据、网格数据与几何数据是否是包含关系。 ■ 新增ST_Within函数，用于判断网格数据与网格数据、网格数据与几何数据是否是被包含关系。 ○ 新增支持矢量金字塔及快速显示的功能： ■ 新增ST_BuildPyramid函数，用于创建矢量金字塔。 ■ 新增ST_DeletePyramid函数，用于删除矢量金字塔。 ■ 新增ST_Tile函数，用于获取MVT格式的矢量瓦片数据。 • Bug修复 ○ 修复在某些情况下创建金字塔会出现内存耗尽的问题。 ○ 修复移动对象无法创建“2000-01-01”时间点的问题。 ○ 修复某些场景下移动对象使用ST_Intersection返回子轨迹错误的问题。

小版本	说明

2.9	• 新特性
	  ○ 新增支持COG（Cloud Optimize Geotiff）文件格式，支持读取COG文件格式中存储的金字塔信息。
	  ○ 新增ST_AddZ函数，支持通过栅格数据的像素值为几何对象添加Z值。
	  ○ 栅格对象空间范围信息获取增强，支持基于金字塔层级查询：
	  ■ 新增ST_Extent函数，用于获得栅格对象的空间范围，以BOX形式返回。
	  ■ 新增ST_Envelope函数，用于获得栅格对象的空间范围，以几何对象形式返回。
	  ■ 新增ST_ConvexHull函数，用于获得栅格对象的空间范围，以几何对象形式返回。
	  ■ 新增ST_Height函数，用于获得栅格对象的像素高度。
	  ■ 新增ST_Width函数，用于获得栅格对象的像素宽度。
	  ■ 修改ST_XMin函数，用于获得栅格对象的X最小值。
	  ■ 修改ST_YMin函数，用于获得栅格对象的Y最小值。
	  ■ 修改ST_XMax函数，用于获得栅格对象的X最大值。
	  ■ 修改ST_YMax函数，用于获得栅格对象的Y最大值。
	• Bug修复
	  ○ 修复外部栅格数据会使用1 x n分块导致性能局限性的问题，允许用户通过存储选项自定义分块的大小。
	  ○ 修复ST_Values函数在查询某些方向的线对象时结果与坐标排序不一致的问题。
	  ○ 修复ST_BestPyramidLevel函数在某些情况下会返回负数的问题。
	  ○ 修复ST_BuildPyramid函数在某些情况下会重复创建金字塔的问题。
	  ○ 修复Truncate栅格表时未能清理对应的块表的问题。
	  ○ 修复ST_ExportTo函数对于CreateOption在某些情况下无效的问题。
	  ○ 修复ST_ClearChunks函数对于表名存在大小写时会出现错误的问题。
	  ○ 修复外部金字塔在某些情况下无法创建overview的问题。
	  ○ 修复具有外部金字塔的栅格对象无法创建内部金字塔的问题。
	  ○ 修复具有NaN数值的栅格数据在计算统计信息时会导致结果不正确的问题。

小版本	说明

2.8	• 新特性
	  ○ 栅格数据元数据访问接口增强：
	  ■ 新增ST_XMin函数，用于获取栅格数据X方向最小值。
	  ■ 新增ST_YMin函数，用于获取栅格数据Y方向最小值。
	  ■ 新增ST_XMax函数，用于获取栅格数据X方向最大值。
	  ■ 新增ST_YMax函数，用于获取栅格数据Y方向最大值。
	  ■ 新增ST_ChunkHeight函数，用于获取栅格数据分块高度。
	  ■ 新增ST_ChunkWidth函数，用于获取栅格数据分块宽度。
	  ■ 新增ST_ChunkBands函数，用于获取栅格数据分块波段数量。
	  ○ 新增ST_SrFromEsriWkt函数，用于支持Esri格式空间参考字符串转换为OGC格式空间参考字符串。
	  ○ 新增栅格数据类型支持Zstd和Snappy压缩方式。
	  ○ 新增点云数据类型支持二进制拷贝功能。
	  ○ 新增支持PROJ_LIB和GDAL_DATA环境变量设置，同时部署相关数据。
	• Bug修复
	  ○ 修复OSS路径非法导致数据库崩溃问题。
	  ○ 修复部分栅格数据导入SRID与定义不一致的问题。

小版本	说明

2.7	• 新特性
	  ○ 新增空间栅格对象的MD5操作函数，可以用于数据的一致性检查和去重等操作：
	  ■ 新增ST_MD5Sum函数，用于获取栅格对象的MD5码值。
	  ■ 新增ST_SetMD5Sum函数，用于设置栅格对象的MD5码值。
	  ○ 新增空间栅格对象OSS认证方式操作函数：
	  ■ 新增ST_AKId函数，用于获取以OSS方式存储的栅格对象的AcessKey ID。
	  ■ 新增ST_SetAccessKey函数，用于设置以OSS方式存储的栅格对象的AcessKey ID和AcessKey Secret。
	  ■ 新增ST_SetAKId函数，用于设置以OSS方式存储的栅格对象的AcessKey ID。
	  ■ 新增ST_SetAKSecret函数，用于设置以OSS方式存储的栅格对象的AcessKey Secret。
	  ○ 新增空间栅格元数据操作函数：
	  ■ 新增ST_ScaleX函数，用于获取栅格对象在空间参考系下X方向像素宽度。
	  ■ 新增ST_ScaleY函数，用于获取栅格对象在空间参考系下Y方向像素宽度。
	  ■ 新增ST_SetScale函数，用于设置栅格对象在空间参考系下像素宽度。
	  ■ 新增ST_SkewX函数，用于获取栅格对象在空间参考系下X方向旋转。
	  ■ 新增ST_SkewY函数，用于获取栅格对象在空间参考系下Y方向旋转。
	  ■ 新增ST_SetSkew函数，用于设置栅格对象在空间参考系下旋转。
	  ■ 新增ST_UpperLeftX函数，用于获取栅格对象在空间参考系下左上角点的X坐标。
	  ■ 新增ST_UpperLeftY函数，用于获取栅格对象在空间参考系下左上角点的Y坐标。
	  ■ 新增ST_SetUpperLeft函数，用于获取栅格对象在空间参考系下左上角点坐标。
	  ■ 新增ST_PixelWidth函数，用于获取栅格对象在空间参考系下像素宽度。
	  ■ 新增ST_PixelHeight函数，用于获取栅格对象在空间参考系下像素高度。
	• Bug修复
	  修复由于聚集函数会导致扩展升级失败的问题。

2.6	• 新特性
	  新增ST_Clip函数，支持基于象元坐标进行裁剪。
	• Bug修复
	  ○ 修复ST_NearestApproachDistance函数名称不正确的问题。
	  ○ 修复ST_MosaicFrom函数在某些情况下崩溃的问题。

3.模型

3.1. 几何模型

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

概述

几何模型完全兼容PostGIS接口，支持已有应用的平滑迁移。

快速入门

- 创建扩展

```
--创建几何扩展
Create extension ganos_geometry cascade;
--创建几何拓扑扩展
Create extension ganos_geometry_topology;
--创建sfcgal插件扩展
Create extension ganos_geometry_sfcgal;
```

- 创建几何表

```
--方式一：直接创建带geometry字段的表
CREATE TABLE ROADS ( ID int4, ROAD_NAME varchar(25), geom geometry(LINESTRING,3857) );
--方式二：先创建普通表，再附加几何字段
CREATE TABLE ROADS ( ID int4, ROAD_NAME varchar(25) );
SELECT AddGeometryColumn( 'roads', 'geom', 3857, 'LINESTRING', 2);
```

- 添加几何约束

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

- 导入几何数据

```
INSERT INTO roads (id, geom, road_name)
 VALUES (1,ST_GeomFromText('LINESTRING(191232 243118,191108 243242)',3857),'北五环');
INSERT INTO roads (id, geom, road_name)
 VALUES (2,ST_GeomFromText('LINESTRING(189141 244158,189265 244817)',3857),'东五环');
INSERT INTO roads (id, geom, road_name)
 VALUES (3,ST_GeomFromText('LINESTRING(192783 228138,192612 229814)',3857),'南五环');
INSERT INTO roads (id, geom, road_name)
 VALUES (4,ST_GeomFromText('LINESTRING(189412 252431,189631 259122)',3857),'西五环');
INSERT INTO roads (id, geom, road_name)
 VALUES (5,ST_GeomFromText('LINESTRING(190131 224148,190871 228134)',3857),'东长安街');
INSERT INTO roads (id, geom, road_name)
 VALUES (6,ST_GeomFromText('LINESTRING(198231 263418,198213 268322)',3857),'西长安街');
```

- 查询几何对象信息

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

- 创建索引

```
--GiST索引
CREATE INDEX [indexname] ON [tablename] USING GIST ( [geometryfield] );
CREATE INDEX [indexname] ON [tablename] USING GIST ([geometryfield] gist_geometry_ops_nd);
VACUUM ANALYZE [table_name] [(column_name)];
--举例
Create INDEX sp_geom_index ON ROADS USING GIST(geom);
VACUUM ANALYZE ROADS (geom);
--创建BRIN索引
CREATE INDEX [indexname] ON [tablename] USING BRIN ( [geometryfield] );
CREATE INDEX [indexname] ON [tablename] USING BRIN ([geometryfield] brin_geometry_inclusion_ops_3
d);
CREATE INDEX [indexname] ON [tablename] USING BRIN ([geometryfield] brin_geometry_inclusion_ops_4
d);
--创建指定大小的brin索引
CREATE INDEX [indexname] ON [tablename] USING BRIN ( [geometryfield] ) WITH (pages_per_range = [num
ber]);
```

- 几何对象存取

```
---判断空间几何对象是否是简单要素类型
 SELECT ST_IsSimple(ST_GeomFromText('POLYGON((1 2, 3 4, 5 6, 1 2))'));
 st_issimple
-------------
 t
(1 row)
 SELECT ST_IsSimple(ST_GeomFromText('LINESTRING(1 1,2 2,2 3.5,1 3,1 2,2 1)'));
 st_issimple
-------------
 f
(1 row)
--查询地形中拥有环岛且面积最大的城市
SELECT gid, name, ST_Area(the_geom) AS area
FROM bc_municipality
WHERE ST_NRings(the_geom) > 1
ORDER BY area DESC LIMIT 1;
gid  | name       | area
-----+--------------+------------------
12   | 安宁市       | 257374619.430216
(1 row)
```

- 空间测量、空间分析和空间关系判断

```
--创建表bc_roads
Create table bc_roads (gid serial, name varchar, the_geom geometry);
--创建表bc_municipality
Create table bc_municipality(gid serial, code integer, name varchar, the_geom geometry);
--长度计算
SELECT sum(ST_Length(the_geom))/1000 AS km_roads FROM bc_roads;
km_roads
------------------
70842.1243039643
(1 row)
--面积计算
SELECT ST_Area(the_geom)/10000 AS hectares FROM bc_municipality WHERE name = 'PRINCE GEORGE';
hectares
------------------
32657.9103824927
(1 row)
--使用ST_Contains函数
SELECT  m.name, sum(ST_Length(r.the_geom))/1000 as roads_km
FROM
 bc_roads AS r, bc_municipality AS m
WHERE
 ST_Contains(m.the_geom,r.the_geom)
GROUP BY m.name
ORDER BY roads_km;
name                | roads_km
----------------------------+------------------
SURREY              | 1539.47553551242
VANCOUVER           | 1450.33093486576
LANGLEY DISTRICT    | 833.793392535662
BURNABY             | 773.769091404338
PRINCE GEORGE       | 694.37554369147
--使用ST_Covers函数
SELECT ST_Covers(smallc,smallc) As smallinsmall,
   ST_Covers(smallc, bigc) As smallcoversbig,
   ST_Covers(bigc, ST_ExteriorRing(bigc)) As bigcoversexterior,
   ST_Contains(bigc, ST_ExteriorRing(bigc)) As bigcontainsexterior
FROM (SELECT ST_Buffer(ST_GeomFromText('POINT(1 2)'), 10) As smallc,
   ST_Buffer(ST_GeomFromText('POINT(1 2)'), 20) As bigc) As foo;
   --Result
 smallinsmall | smallcoversbig | bigcoversexterior | bigcontainsexterior
--------------+----------------+-------------------+---------------------
 t            | f              | t                 | f
(1 row)
--使用ST_Disjoint函数
SELECT ST_Disjoint('POINT(0 0)'::geometry, 'LINESTRING ( 2 0, 0 2 )'::geometry);
 st_disjoint
---------------
 t
(1 row)
SELECT ST_Disjoint('POINT(0 0)'::geometry, 'LINESTRING ( 0 0, 0 2 )'::geometry);
 st_disjoint
---------------
```

```
 t
(1 row)
--使用ST_Overlaps函数
SELECT ST_Overlaps(a,b) As a_overlap_b,
   ST_Crosses(a,b) As a_crosses_b,
      ST_Intersects(a, b) As a_intersects_b, ST_Contains(b,a) As b_contains_a
FROM (SELECT ST_GeomFromText('POINT(1 0.5)') As a, ST_GeomFromText('LINESTRING(1 0, 1 1, 3 5)')  As
b)
   As foo
a_overlap_b | a_crosses_b | a_intersects_b | b_contains_a
------------+-------------+----------------+--------------
f       | f       | t          | t
--使用ST_Relate函数
SELECT ST_Relate(ST_GeometryFromText('POINT(1 2)'), ST_Buffer(ST_GeometryFromText('POINT(1 2)'),
2), '0FFFFF212');
st_relate
-----------
t
--使用ST_Touches函数
SELECT ST_Touches('LINESTRING(0 0, 1 1, 0 2)'::geometry, 'POINT(1 1)'::geometry);
 st_touches
------------
 f
(1 row)
SELECT ST_Touches('LINESTRING(0 0, 1 1, 0 2)'::geometry, 'POINT(0 2)'::geometry);
 st_touches
------------
 t
(1 row)
--使用ST_Within函数
SELECT ST_Within(smallc,smallc) As smallinsmall,
   ST_Within(smallc, bigc) As smallinbig,
   ST_Within(bigc,smallc) As biginsmall,
   ST_Within(ST_Union(smallc, bigc), bigc) as unioninbig,
   ST_Within(bigc, ST_Union(smallc, bigc)) as biginunion,
   ST_Equals(bigc, ST_Union(smallc, bigc)) as bigisunion
FROM
(
SELECT ST_Buffer(ST_GeomFromText('POINT(50 50)'), 20) As smallc,
   ST_Buffer(ST_GeomFromText('POINT(50 50)'), 40) As bigc) As foo;
--Result
 smallinsmall | smallinbig | biginsmall | unioninbig | biginunion | bigisunion
--------------+------------+------------+------------+------------+------------
 t        | t       | f       | t       | t       | t
(1 row)
```

- 删除扩展

```
Drop extension ganos_geometry cascade;
```

SQL参考

详细SQL手册请参见PostGIS官方手册。

3.2. 栅格模型

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

概述

栅格数据可以是数字航空像片、卫星影像、数字图片或甚至扫描的地图。

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

快速入门

- 创建扩展

```
Create Extension Ganos_Raster cascade;
```

- 创建栅格表

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

- 从OSS中导入栅格数据

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

- 查询栅格对象信息

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

- 创建金字塔

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

- 根据视口的世界坐标范围，长和宽来计算最佳的金字塔层级

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

- 获取栅格指定范围的像素矩阵

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

- 计算裁剪区域的像素坐标范围

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

- GPU加速

  当用户购买包含GPU的RDS实例时，自动开启GPU并行加速计算，用户无需设置额外的参数。以创建金字塔为例，介绍支持GPU加速计算的接口ST_BuildPyramid：

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

关于GPU的更多介绍请参见开启GPU加速计算。

- 删除扩展

```
Drop Extension Ganos_raster cascade;
```

SQL参考

详细SQL手册请参见Raster SQL参考。

3.3. 路径模型

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

概述

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

快速入门

- 创建扩展

```
Create Extension Ganos_Networking cascade;
```

- 创建表

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

- 插入记录

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

- 更新表属性

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

- 创建拓扑

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

- 查询最短路径

```
-- 使用dijkstra算法查询最短路径
SELECT * FROM pgr_dijkstra(
  'SELECT id, source, target, cost, reverse_cost FROM edge_table',
  2, 3
);
 seq | path_seq | node | edge | cost | agg_cost
-----+----------+------+------+------+----------
  1 |     1 |  2 |  4 |  1 |     0
  2 |     2 |  5 |  8 |  1 |     1
  3 |     3 |  6 |  9 |  1 |     2
  4 |     4 |  9 | 16 |  1 |     3
  5 |     5 |  4 |  3 |  1 |     4
  6 |     6 |  3 | -1 |  0 |     5
(6 rows)
-- 使用astar算法查询最短路径
SELECT * FROM pgr_astar(
  'SELECT id, source, target, cost, reverse_cost, x1, y1, x2, y2 FROM edge_table',
  2, 12,
  directed := false, heuristic := 2);
 seq | path_seq | node | edge | cost | agg_cost
-----+----------+------+------+------+----------
  1 |     1 |  2 |  2 |  1 |     0
  2 |     2 |  3 |  3 |  1 |     1
  3 |     3 |  4 | 16 |  1 |     2
  4 |     4 |  9 | 15 |  1 |     3
  5 |     5 | 12 | -1 |  0 |     4
(5 rows)
-- 使用trsp 路径算法
SELECT * FROM pgr_trsp(
    'SELECT id::INTEGER, source::INTEGER, target::INTEGER, cost FROM edge_table',
    2, 7, false, false,
    'SELECT to_cost, target_id::int4,
    from_edge || coalesce('','' || via_path, '''') AS via_path
    FROM restrictions'
  );
 seq | id1 | id2 | cost
-----+-----+-----+------
  0 |  2 |  4 |  1
  1 |  5 | 10 |  1
  2 | 10 | 12 |  1
  3 | 11 | 11 |  1
  4 |  6 |  8 |  1
  5 |  5 |  7 |  1
  6 |  8 |  6 |  1
  7 |  7 | -1 |  0
(8 rows)
```

- 删除扩展

```
Drop Extension Ganos_Networking cascade;
```

SQL参考

详细SQL手册请参见pgrouting 官方文档 。

3.4. 点云模型

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

概述

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

点云数据类型

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

点云元数据

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

快速入门

- 创建扩展

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

- 插入点云schema

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

- 创建点云表

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

- 插入pcpoint类型数据

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

- 插入pcpatch类型数据

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

- pcpatch属性平均值计算

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

- 删除扩展

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

SQL参考

详细SQL手册请参见 PointCloud SQL参考。

3.5. 轨迹模型

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

概述

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

重要说明

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

快速入门

- 创建扩展

```
Create Extension Ganos_trajectory cascade;
```

- 轨迹的枚举类型

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

- 创建轨迹表

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

- 插入轨迹数据

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

- 创建轨迹空间索引

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

- 创建特定维度轨迹索引

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

- 查询轨迹起、止时间

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

- 轨迹查询

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

- 分析轨迹间的相近性

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

- 删除扩展

```
Drop Extension Ganos_trajectory cascade;
```

SQL参考

详细SQL手册请参见 Trajectory SQL参考。

3.6. 网格模型

网格模型是在GeoSOT地球空间剖分理论的基础上发展出的一种离散化、多尺度区域位置标识体系。

概述

您可以通过网格模型为地球空间（从地心至地上）中的各种对象赋予一个全球唯一编码，任意一个实体对象都可通过此编码在同一区城范围内和各种不同的数据建立内在关联。

快速入门

- 创建扩展

```
CREATE EXTENSION Ganos_GeomGrid CASCADE;
```

- 创建具有网格码的表

```
CREATE TABLE t_grid(id integer,
            geom geometry, -- 几何对象
            grid1 geomgrid[], -- 网格码，精度1
            grid2 geomgrid[], -- 网格码，精度2
            grid3 geomgrid[] -- 网格码，精度3
           );
```

- 计算网格码

```
--向表中插入数据。
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));
--创建不同精度的网格码。
UPDATE t_grid
SET grid1 = ST_AsGrid(geom, 10),
  grid2 = ST_AsGrid(geom, 15),
  grid3 = ST_AsGrid(geom, 26);
```

- 创建网格码索引

```
--针对不同精度的网格码创建GIN索引。
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);
```

- 查询

```
--完全在某个网格中。
SELECT id
FROM t_grid
WHERE grid2 = ARRAY[ST_GridFromText('G001310322230230')];
--和某个网格相交。
SELECT id
FROM t_grid
WHERE grid3 @> ARRAY[ST_GridFromText('G00131032223023031031033223')];
--和某些网格相交。
SELECT id
FROM t_grid
WHERE grid3 && ARRAY[ST_GridFromText('G00131032223023031031211001'),
                ST_GridFromText('G00131032223023031031211111')];
--和某个几何对象在网格上相交。
SELECT id
FROM t_grid
WHERE grid3 &&
ST_AsGrid(
 ST_GeomFromText('LINESTRING(116.31522216796875 39.910277777777778, 116.31522217797875 39.910
277776787778)', 4490), 26);
```

- 删除扩展

```
DROP EXTENSION Ganos_GeomGrid CASCADE;
```

3.7. 矢量金字塔

矢量金字塔是为了能够快速显示大规模空间几何数据（千万级以上）而设计的一种结构。

概述

矢量金字塔对空间几何数据创建稀疏索引，按规则对密集区域预处理，可以输出标准的mvt-pbf格式数据。

通过Ganos提供的矢量金字塔，亿条空间几何记录可以实现分钟级预处理、秒级终端显示。

快速入门

- 创建扩展

```
CREATE EXTENSION ganos_geometry_pyramid CASCADE;
```

- 创建空间表的金字塔

```
--为数据表test创建金字塔，指定表test的要素ID字段名，必须为int4或int8类型。
--指定表test的空间字段名称，需要先为该字段创建空间索引。
SELECT ST_BuildPyramid('test', 'geom', 'id', '');
```

- 读取金字塔中MVT数据

```
--从金字塔中读取瓦片编号为'0_0_0'的数据（任意编号，不管金字塔中是否存在，都可返回数据）。
--瓦片编号方式为: z_x_y，投影坐标系为EPSG:3857。
SELECT ST_Tile('test', '0_0_0');
```

- 删除金字塔

```
--删除名为test的金字塔。
SELECT ST_DeletePyramid('test');
```

- 删除扩展

```
DROP EXTENSION ganos_geometry_pyramid CASCADE;
```

参数设置

- 为金字塔命名

  金字塔默认和数据表同名，您也可以指定金字塔名称，实现一份数据对应多个金字塔。

```
--为test表创建一个名为hello的金字塔。
SELECT ST_BuildPyramid('test', 'geom', 'id', '{"name": "hello"}');
```

- 并行构建

  指定构建矢量金字塔的并行任务数，默认为0（表示并行最大化）。并行任务数最大不应该超过CPU数量的4倍。

  并行构建使用了两阶段事务机制，需要设置数据库max_prepared_transactions参数，需要设置max_prepared_transactions = 100或者更高，重启生效。

```
--使用4个CPU来并行构建矢量金字塔。
SELECT ST_BuildPyramid('test', 'geom', 'id','{"parallel": 4}');
```

- 瓦片参数

  您可以指定瓦片的尺寸、外扩大小。

  - 尺寸取值为0~4096，且必须是256的整数倍。

- 外扩大小取值为0~256。

```
--指定瓦片的大小为512，外扩大小为8。
SELECT ST_BuildPyramid('test', 'geom', 'id','{
              "tileSize": 512,
              "tileExtend": 8
             }');
```

- 金字塔最大层级

  您可以指定金字塔的最大层级（默认为16），当地图的zoom层级大于某级时，就不再使用这个图层或者不需要生成金字塔。如不设置，矢量金字塔会根据数据的密度自动计算出合理的最大层级。

```
--指定金字塔的最大层级为12级，超过12级则会实时读取数据生成mvt。
SELECT ST_BuildPyramid('test', 'geom', 'id', '{"maxLevel": 12}');
```

- 分层处理

  您可以为金字塔的每个层级设置不同的处理条件，例如显示字段、过滤条件等。

  通过添加 `buildRules` 规则，为每个层级设置生成条件。

  顶层金字塔生成耗时较长，可以通过分层规则跳过顶层的处理。

```
--分层处理生成金字塔
--第0层到第5层，不显示任何数据，设置过滤条件为"1!=1"生成空的mvt。
--第6层到第9层，显示code=1的数据，并且包含name字段。
--第10层到第15层，无过滤条件，包含name、width两个字段。
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"]
               }
              }
             ]
            }');
```

高级功能

按规则融合数据：您可以按属性规则对瓦片范围内的数据进行融合，减少数据量。

```
-- 将 code=1、code=2 的数据，分别进行融合处理
SELECT ST_BuildPyramid('test', 'geom', 'id', '{
            "buildRules":[
                {
                  "level":[0,1,2,3,4,5],
                  "value": {
                    "merge": ["code=1","code=2"]
                  }
                }
              ]
            }');
```

4.使用进阶

4.1. 开启时空两阶段查询优化

Ganos在时空索引框架基础上进行的深度优化，可减少时空查询中额外的数据I/O及计算开销。

背景信息

时空数据库传统的查询都是经典的两阶段查询处理方法（粗糙集过滤和精确过滤），首先利用时空多维索引进行粗糙集过滤，筛选出来的中间结果集再经过精确函数判断，得到最终的结果集。

示例

先利用test表上的空间索引与查询对象进行粗过滤，得到中间结果集ID，根据中间结果集ID再取出实际的记录数据，利用_st_intersects进行精确过滤，最终得到符合条件的记录集。

这其中，利用空间索引进行粗糙集过滤后，依旧有大量无关的对象需要进入到精确过滤阶段，一方面带来数据I/O开销，另一方面由于精确过滤计算过于复杂，会带来大量无用计算。

```
select id from test where st_intersects(ST_GeomFromText('POLYGON((250000 -268000, 250000 270000, 280000 270000,280000 -268000, 250000 -268000))', 3857),geom)=true;
```

两阶段查询优化

两阶段查询优化对于时空范围查询、PIP（point in polygon）查询等场景有较大的性能提升，该优化由PolarDB中的guc参数控制，对用户使用透明，当前默认为关闭状态。

在数据库session中开启或关闭查询优化的命令如下：

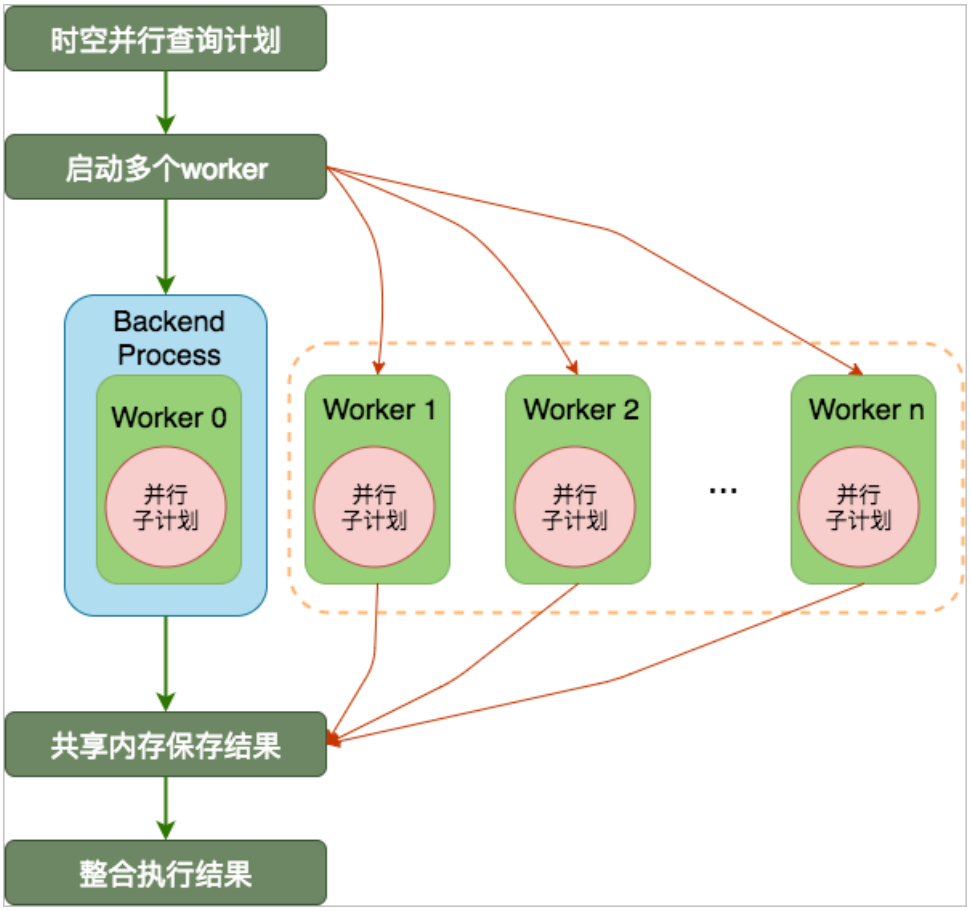
```
#开启
set polar_enable_gist_refine = true;
#关闭
set polar_enable_gist_refine = false;
```

4.2. 开启时空并行查询

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

并行查询原理

PG并行查询是表级的并行，其并行查询示意图如下。



注意事项

- 并行查询的worker数量越大，worker数越多，查询时CPU负载越重，对于CPU负载本身较重的场景建议woker数量设置为2较合适，即max_parallel_workers_per_gather=2。
- 对于服务器内存有限的高并发访问，开启并行查询时，需要控制参数work_men（min 64KB），确保并发访问数量乘以并行worker数量乘以work_mem不超过服务器内存的60%。

使用方法

开启Ganos并行查询的方法如下：

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

     > **说明** “n”代表并行worker数，该值可参见max_parallel_workers_per_gather。

2. 提高Ganos相关函数的cost成本。

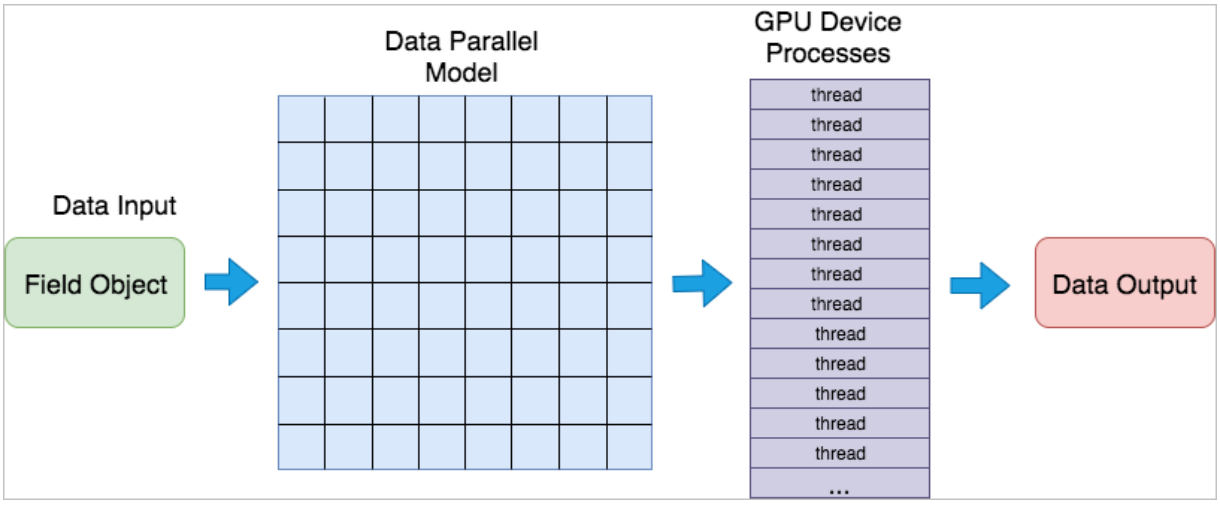
在创建Ganos模块扩展时，通常默认有个函数cost成本，如果表数据量较小，但函数属于计算密集，并且适合开启并行执行，此时默认不会开启并行查询，需要提高函数的cost成本后才能开启并行查询。

4.3. 开启GPU加速计算

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

加速原理

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



注意事项

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

使用方法

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

1. 确认当前环境是否有GPU设备。
   i. 创建ganos_raster扩展 `create extension ganos_raster cascade` 。

ii. 执行SQL语句 `select st_checkgpu()` 。

- 当前环境检测到GPU设备，成功返回GPU设备信息。

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

- 当前环境未检测到GPU设备，则返回错误信息。

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

> **说明** 只有带GPU设备的环境才能启用GPU加速计算。

2. 开启和关闭会话级GPU使用状态。

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

  执行set ganos.raster.use_cuda=off

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

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

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

3. Ganos中实现GPU加速计算的模块。

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

5.Raster SQL参考

5.1. 基本概念

本章节将为您介绍Raster SQL的基本概念。

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

5.2. 并行操作

Ganos支持利用多个CPU提升查询或计算性能，这种特性被称为并行操作。Ganos支持并行执行SQL语句和并行操作raster对象。

并行执行SQL语句

- 原理

  PostgreSQL支持利用多个CPU生成并行的查询计划，并将执行任务分配到多个CPU上以提升性能。

- 适用场景

  SQL语句并行执行非常适合在大量的raster对象中过滤出符合空间范围要求或属性要求的raster对象，能够极大地降低查询时间。

- 适用范围

  所有Ganos Raster的只读函数（例如属性查询）都支持并行查询。

并行操作Raster对象

- 原理

  通常情况下raster对象并行操作技术可以应用于单个raster对象的操作过程，通过多个CPU并行计算raster对象中的子集内容，从而降低整个raster对象的操作时间。每个子集内容的计算均独立运行，当所有子集执行完成，整个raster对象并行操作过程也同时结束。

- 适用场景

  raster非常大，操作耗时很长的场景。

- 适用范围

  目前支持raster对象并行操作的函数包括：

  - ST_ImportFrom
  - ST_BuildPyramid
  - ST_RPCRectify

使用前准备工作

- 设置Prepared状态最大事务数

  通过参数max_prepared_transactions设置Prepared状态最大事务数。该参数默认值为0，建议设置为与max_connections参数一样，确保每个连接都有一个Prepared事务可以处理。

  > **说明** 修改本参数后需要重启实例。

- 设置并行度

  如果支持并行操作的函数未指定并行度或指定并行度为0，则使用GUC（Grand Unified Configuration）参数ganos.parallel.degree的值作为默认的并行度。ganos.parallel.degree参数默认值为1， 表示不支持并行执行。

  您也可以在支持并行操作的函数中指定并行度参数为一个正整数，可以将raster对象的计算任务拆分为并行任务执行。示例如下：

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

- 设置事务一致性

  GUC参数ganos.parallel.transaction用于指定并行事务级别，取值如下：

  - transaction_commit（默认值）：支持并行事务根据主事务进行提交或回滚。
  - fast_commit：不支持并行事务回滚。

> **说明**
> - 您可以使用ST_CreateChunkTable函数预先创建块表以达到最佳性能。
> - 如果并行函数结果不支持匿名临时块表，请预先创建块表并在存储选项中的chunktable参数中指定该块表。

5.3. Raster创建

5.3.1. ST_CreateRast

创建一个基于阿里云对象存储服务（OSS）的raster对象。

语法

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

参数

参数名称	描述
url	OSS影像文件的路径。如果是具有SubSet的NetCDF，可以通过 `:<name>` 方式指定。
storageOption	基于JSON格式的字符串，描述raster对象金字塔的分块存储信息。

storageOption支持的参数如下：

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

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

描述

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

Endpoint为OSS的地域节点。为保证数据导入的性能，请确保云数据库PostgreSQL与OSS所在地域相同，详情请参见OSS Endpoint。

示例

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

5.4. 导入导出

5.4.1. ST_ImportFrom

从一个OSS文件导入到数据库。

语法

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

参数

参数名称	描述
chunkTableName	块表的名称，名称必须符合数据库表名的规范。
url	外部文件路径。详情请请参见ST_CreateRast中构建路径的描述。
storageOption	JSON字符串，用于指定raster对象的存储信息。
importOption	JSON字符串，用于指定导入选项。

storageOption参数说明如下。

参数名称	类型	默认值	说明
chunking	boolean	true	是否使用分块存储。
chunkdim	string	与原始数据一致	分块的维度信息。格式为：(w, h, b)。 **说明** 当chunking=**true**时，本参数才生效。

参数名称	类型	默认值	说明
compression	string	lz4	压缩算法类型。取值如下： • none • jpeg • zlib • png • lzo • lz4 • snappy • zstd • jp2k
quality	integer	75	压缩质量。只针对jpeg和jp2k压缩算法生效。
interleaving	string	与原始数据一致	交错方式。取值如下： • bip：Band interleaved by pixel • bil：Band nterleaved by pixel • bsq：Band Sequential
blockendian	string	'NDR'	块存储字节序。取值如下： • NDR：little endian • XDR：big endian
celltype	string	与原始数据一致	像素类型。取值如下： • 1bb：1 bit • 2bui：2bit unsigned integer • 4bui：4bit unsigned integer • 8bs：8bit signed integer • 8bui：8bit unsigned integer • 16bsi：16bit signed integer • 16bui：16bit unsigned integer • 32bsi：32bit signed integer • 32bui：32bit unsigned integer • 32bf：32bit float • 64bsi：64bit signed integer • 64bui：64bit unsigned integer • 64bf：64bit float

importOption参数说明如下。

参数名称	类型	默认值	说明

参数名称	类型	默认值	说明
mapping_oss_file	boolean	false	是否将OSS上文件映射为内存文件。 **说明** 栅格数据读取时部分驱动未对OSS存储的数据进行优化，会产生大量小数据量请求导致性能较低。此参数可以将OSS完整映射为内存对象从而提升性能。文件总大小由GUC参数ganos.raster.memory_oss_file_max_size设置。
parallel	integer	使用GUC参数ganos.parallel.degree的值。	设置操作并行度。取值范围为1~64。

描述

函数将创建一个raster对象，并将外部OSS文件导入到该对象中。

支持的数据类型请通过ST_RasterDrivers函数获取。

示例

```
-- 仅指定外表名称和路径
Select ST_ImportFrom('chunk_table','OSS://<ak>:<ak_secret>@oss-cn-beijing-internal.aliyuncs.com/mybucket/data/image.tif');
-- 指定具有Subset的NetCDF对应的影像
Select ST_ImportFrom('chunk_table','OSS://<ak>:<ak_secret>@oss-cn-beijing-internal.aliyuncs.com/mybucket/data/image.nc:hcc');
-- 导入时指定分块大小与压缩类型
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"}');
-- 导入时指定并行度
Select ST_ImportFrom('chunk_table','OSS://<ak>:<ak_secret>@oss-cn-beijing-internal.aliyuncs.com/mybucket/data/image.nc:hcc', '{}', '{"parallel": 4}');
```

5.4.2. ST_ExportTo

将一个raster对象导出为OSS文件。

语法

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

参数

参数名称	描述
source	需要导出的raster对象。

参数名称	描述
format	导出的数据，常见如 GTiff，BMP 等。
url	外部文件路径。详情请参见ST_CreateRast 中构建路径的描述。
level	金字塔级别。

描述

导出成功返回true，失败则返回false。

format指定导出格式的名称，常见格式如下。

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

示例

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

5.4.3. ST_AsImage

将栅格对象转化为影像格式二进制流。

语法

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

参数

参数名称	描述
raster_obj	需要计算的raster对象。
extent	影像的范围，默认使用地理坐标系统。
pyramidLevel	影像金字塔层级，从0开始，默认值为0。
bands	需要获取的波段列表，从0开始，用 '0-2' 或者 ‘1,2,3’ 这种形式表示。默认为空。JPEG为1或3，PNG为1、2、3或4。默认使用前三个波段。
format	输出影像格式，取值如下： • PNG • JPEG
option	JSON字符串类型的转换选项。

option参数说明如下。

参数名称	描述	类型	默认值	说明
nodata	是否使用nodata值。	bool	false	• true：需要处理nodata值。 • false：nodata值作为普通值处理。
nodataValue	nodata值。	integer	0	当nodata=true时，需为nodata设置参数值。
rast_coord	传入的box是否为像元坐标。	bool	false	如果是像元坐标，横坐标x表示像元的列号（起始为0），纵坐标y表示像元的行号（起始为0）。
strength	是否进行增强。	string	none	显示增强的方式，取值： • none：不进行增强。 • stats：使用统计值进行拉伸。
quality	压缩质量。	integer	75	压缩质量，取值为1~100。

描述

- 函数将返回一个bytea表示的影像格式。
- 默认的裁剪缓存为100 MB，代表最多只能裁剪出100 MB大小的结果数据，如果需要调整返回结果大小，可使用参数ganos.raster.clip_max_buffer_size设置缓存的大小。
- 波段数量说明如下：
  - 1：表示灰度影像。
  - 2：表示灰度和Alpha波段。
  - 3：表示R波段、G波段和B波段。

- 4：表示R波段、G波段、B波段和Alpha波段。

示例

```
--使用裁剪缓存范围。
SELECT ST_AsImage(raster_obj,
          '(-180,-90), (0,0)'::Box)
FROM raster_table
WHERE id=1;
--指定金字塔层级。
SELECT ST_AsImage(raster_obj,
          '(-180,-90), (0,0)'::Box,
          1)
FROM raster_table
WHERE id=1;
--指定波段使用裁剪缓存范围。
SELECT ST_AsImage(raster_obj,
          '(-180,-90), (0,0)'::Box,
          1,
          '0-2')
FROM raster_table
WHERE id=1;
--指定压缩格式。
SELECT ST_AsImage(raster_obj,
          '(-180,-90), (0,0)'::Box,
          1,
          '0-2',
          'PNG')
FROM raster_table
WHERE id=1;
--使用统计值拉伸。
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;
--指定象元坐标裁剪缓存并使用统计值拉伸。
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;
```

5.4.4. ST_AsPNG

将栅格对象转化为PNG格式二进制流。

语法

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

参数

参数名称	描述
raster_obj	需要计算的raster对象。
extent	影像的范围，默认使用地理坐标系统。
pyramidLevel	影像金字塔层级，从0开始，默认值为0。
bands	需要获取的波段列表，从0开始，用 '0-2' 或者 '1,2,3' 这种形式表示。默认为空。PNG波段数量为1、2、3或4。默认使用前三个波段。
option	JSON字符串类型的转换选项。

option参数说明如下。

参数名称	描述	类型	默认值	说明
nodata	是否使用nodata值。	bool	false	• true：需要处理nodata值。 • false：nodata值作为普通值处理。
nodataValue	nodata值。	integer	0	在nodata=true时，需为nodata设置参数值。
rast_coord	传入的box是否为像元坐标。	bool	false	如果是像元坐标，横坐标x表示像元的列号（起始为0），纵坐标y表示像元的行号（起始为0）。
strength	是否进行增强。	string	none	显示增强的方式，取值： • none：不进行增强。 • stats：使用统计值进行拉伸。
quality	压缩质量。	integer	75	压缩质量，取值为1~100。

描述

- 函数将返回一个bytea表示的影像格式。
- 默认的裁剪缓存为100 MB，代表最多只能裁剪出100 MB大小的结果数据，如果需要调整返回结果大小，可使用参数ganos.raster.clip_max_buffer_size设置缓存的大小。
- 波段数量说明如下：
  - 1：表示灰度影像。

- 2：表示灰度和Alpha波段。
- 3：表示R波段、G波段和B波段。
- 4：表示R波段、G波段、B波段和Alpha波段。

示例

```
--使用裁剪范围。
SELECT ST_AsPNG(raster_obj,
            '(-180,-90), (0,0)'::Box)
FROM raster_table
WHERE id =1;
--指定金字塔层级。
SELECT ST_AsPNG(raster_obj,
            '(-180,-90), (0,0)'::Box,
            1)
FROM raster_table
WHERE id =1;
--指定波段使用裁剪范围。
SELECT ST_AsPNG(raster_obj,
            '(-180,-90), (0,0)'::Box,
            1,
            '0-2')
FROM raster_table
WHERE id =1;
-- 使用统计值拉伸。
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;
```

5.4.5. ST_AsJPEG

将栅格对象转化为JPEG格式二进制流。

语法

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

参数

参数名称	描述
raster_obj	需要计算的raster对象。

参数名称	描述
extent	影像的范围，默认使用地理坐标系统。
pyramidLevel	影像金字塔层级，从0开始，默认值为0。
bands	需要获取的波段列表，从0开始，用 '0-2' 或者 ‘1,2,3’ 这种形式表示。默认为空。JPEG波段数量为1、2、3或4。默认使用前三个波段。
option	JSON字符串类型的转换选项。

option参数说明如下。

参数名称	描述	类型	默认值	说明
nodata	是否使用nodata值。	bool	false	• true：需要处理nodata值。 • false：nodata值作为普通值处理。
nodataValue	nodata值。	integer	0	在nodata=true时，需为nodata设置参数值。
rast_coord	传入的box是否为像元坐标。	bool	false	如果是像元坐标，横坐标x表示像元的列号（起始为0），纵坐标y表示像元的行号（起始为0）。
strength	是否进行增强。	string	none	显示增强的方式，取值： • none：不进行增强。 • stats：使用统计值进行拉伸。
quality	压缩质量。	integer	75	压缩质量，取值为1~100。

描述

- 函数将返回一个bytea表示的影像格式。
- 默认的裁剪缓存为100 MB，代表最多只能裁剪出100 MB大小的结果数据，如果需要调整返回结果大小，可使用参数ganos.raster.clip_max_buffer_size设置缓存的大小。
- 波段数量说明如下：
  - 1：表示灰度影像。
  - 3：表示R波段、G波段和B波段。

示例

```
--使用裁剪范围。
SELECT ST_AsJPEG(raster_obj,
            '(-180,-90), (0,0)'::Box)
FROM raster_table
WHERE id =1;
--指定金字塔层级。
SELECT ST_AsJPEG(raster_obj,
            '(-180,-90), (0,0)'::Box,
            1)
FROM raster_table
WHERE id =1;
--指定波段使用裁剪范围。
SELECT ST_AsJPEG(raster_obj,
            '(-180,-90), (0,0)'::Box,
            1,
            '0-2')
FROM raster_table
WHERE id =1;
--使用统计值拉伸。
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;
```

5.4.6. ST_AsDatasetFile

将栅格对象转化为文件格式二进制流。

语法

```
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);
```

参数

参数名称	描述
raster_obj	需要计算的raster对象。
extent	影像的范围，默认使用地理坐标系统。

参数名称	描述
pyramidLevel	影像金字塔层级，从0开始，默认值为0。
bands	需要获取的波段列表，从0开始，用 '0-2' 或者 '1,2,3' 这种形式表示。默认为 '' ，获取所有波段。
format	输出影像文件格式。详情请参见ST_RasterDrivers函数。
create_option	JSON字符串类型的数据集创建选项。详情请参见ST_RasterDrivers函数。
process_option	JSON字符串类型的操作选项。 参数为rast_coord，表示传入的box是否为像元坐标。 如果是像元坐标，横坐标x表示像元的列号（起始为0），纵坐标y表示像元的行号（起始为0）。
ext	文件后缀名称，常见如TIF，XML等。
data	数据文件二进制流数据。

描述

- 文件格式必须在ST_RasterDrivers函数中，对应驱动can_asfile字段为true。
- 默认的裁剪缓存为100 MB，代表最多只能裁剪出100 MB大小的结果数据，如果需要调整返回结果大小，可使用参数ganos.raster.clip_max_buffer_size设置缓存的大小。
- 参数create_option可以通过ST_RasterDrivers函数中的create_options获得。

示例

```
--使用裁剪范围。
SELECT ST_AsDatasetFile(raster_obj,
            '(-180,-90), (0,0)'::Box)
FROM raster_table
WHERE id =1;
--指定金字塔层级。
SELECT ST_AsDatasetFile(raster_obj,
            '(-180,-90), (0,0)'::Box,
            1)
FROM raster_table
WHERE id =1;
--指定波段使用裁剪范围。
SELECT ST_AsDatasetFile(raster_obj,
            '(-180,-90), (0,0)'::Box,
            1,
            '0-2')
FROM raster_table
WHERE id =1;
--指定格式为GIF。
SELECT ST_AsDatasetFile(raster_obj,
            '(-180,-90), (0,0)'::Box,
            1,
            '0-2',
               'GIF')
FROM raster_table
WHERE id =1;
--指定格式创建选项。
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;
--指定使用像元坐标。
SELECT ST_AsDatasetFile(raster_obj,
            '(0,0), (100,100)'::Box,
            1,
            '0-2',
               'GTiff',
               '{}',
               '{"rast_coord":"true"}')
FROM raster_table
WHERE id =1;
```

5.5. 金字塔操作

5.5.1. ST_BuildPyramid

创建影像金字塔。

语法

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

参数

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

storageOption参数说明如下。

参数名称	类型	说明
chunkdim	string	分块的维度信息，格式为 `(w, h, b)` ，默认从原始影像中读取分块大小。
interleaving	string	交错方式，取值如下： • bip：像素交错 • bil：行交错 • bsq（默认值）：波段交错 • auto：根据原始影像自动选择

参数名称	类型	说明
compression	string	压缩算法类型，取值如下： • none • jpeg • zlib • png • lzo • lz4（默认值） • zstd • snappy • jp2k
quality	integer	压缩质量。只针对jpeg和jp2k压缩算法生效，默认值为75。

描述

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

示例

```
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;
--使用jpeg2000压缩
--确保指定所有波段在一个分块内
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;
-- 使用并行方式创建
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;
```

5.5.2. ST_deletePyramid

删除影像金字塔。

语法

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

参数

参数名称	描述
source	需要删除金字塔的raster对象。

描述

删除影像金字塔，重置影像元数据，删除金字塔块数据。

示例

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

5.5.3. ST_BestPyramidLevel

根据视口的世界坐标范围，长和宽来计算最佳的金字塔层级。

语法

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

参数

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

描述

raster对象必须要有完整的空间参考信息（srid值有效）。

示例

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

5.6. 坐标系统转换

5.6.1. ST_Rast2WorldCoord

由像元坐标及像元所在金字塔层级，根据仿射变换公式计算世界坐标。

语法

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

参数

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

描述

raster对象必须要有完整的空间参考信息。

示例

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

5.6.2. ST_World2RastCoord

由世界坐标及像元所在金字塔层级，根据逆仿射变换公式计算像元坐标。

语法

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

参数

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

描述

raster对象必须要有完整的空间参考信息。

返回的几何对象，横坐标x值表示象元的列号，纵坐标y值表示象元的行号。

示例

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

5.7. 像素值操作

5.7.1. ST_ClipDimension

计算Clip结果的像素坐标。

语法

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

参数

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

描述

raster对象必须要有完整的空间参考信息（srid值有效）。

示例

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

5.7.2. ST_Clip

对raster对象进行裁剪操作。

语法

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

参数

参数名称	描述
raster_obj	需要裁剪的raster对象。
pyramidLevel	金字塔层级。
extent	需要裁剪的范围，格式为 '((minX,minY),(maxX,maxY))' 。
boxType	范围的类型，只能是以下一种： • Raster （像元坐标） • World （世界坐标）
destSrid	指定输出像元子集的空间参考值。
geometry	需要裁剪的geometry对象。
bands	需要裁剪的波段，用 '0-2' 或者 ‘1,2,3’ 这种形式表示，以0开始。 默认为 '' ，表示裁剪所有的波段。
nodata	用float8[]表示的nodata数值。 如果数值个数少于波段数量，则使用波段设置的nodata值填充。如果波段未设置nodata，则用0填充。
clipOption	表示裁剪选项。
storageOption	表示返回结果的存储选项。

clipOption参数如下。

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

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

storageOption参数如下。

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

描述

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

示例

```
Select ST_Clip(raster_obj, 0, '((128.980,30.0),(129.0,30.2))', 'World');
Select ST_Clip(raster_obj, 0, '((128.980,30.0),(129.0,30.2))', 'World', 4326);
-- 使用geometry裁剪
-- 都是用默认值裁剪
SELECT (ST_CLIP(rast, ST_geomfromtext('Polygon((0 0, 45 45, 90 45, 45 0, 0 0))', 4326), 0)).* from clip_table w
here id =1
-- 使用白色作为背景色填充并且压缩为png图片
SELECT (ST_CLIP(rast, ST_geomfromtext('Polygon((0 0, 45 45, 90 45, 45 0, 0 0))', 4326), 0, '', ARRAY[254,254,25
4], '', '{"compression":"png","interleaving":"bip"}')).* from clip_table where id =1;
-- 使用geometry的窗口裁剪
SELECT (ST_CLIP(rast, ST_geomfromtext('Polygon((0 0, 45 45, 90 45, 45 0, 0 0))', 4326), 0, '', NULL, '{"window_
clip":true}', '')).* from clip_table where id =1;
```

5.7.3. ST_ClipToRast

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

语法

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

参数

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

clipOption参数如下。

参数名称	类型	默认值	描述
window_clip	bool	false	是否使用Geometry的外包框进行裁剪。取值： • true：使用Geometry的MBR裁剪。 • false：使用Geometry对象裁剪。
rast_coord	bool	false	传入的Geometry是否使用的是象元坐标。如果是象元坐标，横坐标x表示象元的列号，纵坐标y表示象元的行号。

storageOption参数如下。

参数名称	类型	默认值	描述
chunking	boolean	和原始Raster一致	是否使用分块存储。

参数名称	类型	默认值	描述
chunkdim	string	和原始Raster一致	分块的维度信息。在chunking=true时才有效。
chunktable	string	''	分块表名称。如果传入 '' 值，则会产生一个随机表名临时块表用于存放数据。 该临时表只在当前会话中有效。如果需要保持一个可访问的裁剪对象，则需要指定块表名称。
compression	string	lz4	压缩算法类型。取值： • none • jpeg • zlib • png • lzo • lz4
quality	integer	75	压缩质量，只针对jpeg压缩算法。
interleaving	string	和原始Raster一致	交错方式。取值： • bip：波段按像元交叉 • bil：波段按行交叉 • bsq：波段顺序
endian	string	和原始Raster一致	字节序。取值： • NDR：小字节序（Little endian） • XDR：大字节序（Big endian）

描述

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

示例

```
-- 永久表
CREATE TEMP TABLE rast_clip_result(id integer, rast raster);
-- 临时表
CREATE TEMP TABLE rast_clip_result_temp(id integer, rast raster);
-- 默认裁剪并存放到临时表中
INSERT INTO rast_clip_result_temp(id, rast)
select 1, ST_ClipToRast(rast, ST_geomfromtext('Polygon((0 0, 45 45, 90 45, 45 0, 0 0))', 4326), 0)
from clip_table
where id =1;
-- 使用白色作为背景色填充，并保存到永久表中
INSERT INTO rast_clip_result(id, rast)
select 2, ST_ClipToRast(rast, ST_geomfromtext('Polygon((0 0, 45 45, 90 45, 45 0, 0 0))', 4326), 0, '', ARRAY[254,
254,254], '', '{"chunktable":"clip_rbt"}')
from clip_table
where id =1;
-- 使用Geometry的窗口裁剪
INSERT INTO rast_clip_result_temp(id, rast)
SELECT 3, ST_ClipToRast(rast, ST_geomfromtext('Polygon((0 0, 45 45, 90 45, 45 0, 0 0))', 4326), 0, '', NULL, '{"
window_clip":true}', '')
from clip_table
where id =1;
```

5.7.4. ST_Values

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

语法

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

参数

参数名称	描述
raster_obj	需要裁剪的raster对象。
pyramidLevel	金字塔层级。
geometry	需要裁剪的geometry对象。
bands	需要裁剪的波段，用 '0-2' 或者 ‘1,2,3’ 这种形式表示，以0开始。 默认为 '' ，表示裁剪所有的波段。
clipOption	json字符串表示的裁剪选项。

参数名称	描述
point	返回结果字段之一，像素值的经纬度（地理）坐标。
band	返回结果字段之一，像素值所在波段号。
value	返回结果字段之一，像素值。

clipOption参数如下。

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

描述

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

示例

```
-- 基于点选择
SELECT ( ST_Values(rast, ST_geomfromtext('POINT(128.135 29.774)', 4326)::geometry, 0,'{"window_clip":"true"}')).*
from rat_clip WHERE id=1;
   coord    | band | value
--------------+------+-------
 (127.8,29.7) |   0 |   11
 (127.8,29.7) |   1 |   10
 (127.8,29.7) |   2 |   50
(3 rows)
-- 基于线选择
SELECT ( ST_Values(rast, ST_geomfromtext('LINESTRING(0 0, 45 45, 90 0, 135 45)', 4326)::geometry, 0)).*
from rat_clip WHERE id=1 limit 10;
   coord    | band | value
-------------+------+-------
 (44.1,45)  |   0 |  115
 (44.1,45)  |   1 |  112
 (44.1,45)  |   2 |   69
 (45,45)    |   0 |  122
 (45,45)    |   1 |  117
 (45,45)    |   2 |   75
 (134.1,45) |   0 |   37
 (134.1,45) |   1 |   64
 (134.1,45) |   2 |   13
 (43.2,44.1) |   0 |   66
(10 rows)
```

5.7.5. ST_AddZ

根据栅格的波段值设置geometry的z值。

语法

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

参数

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

描述

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

> **说明** 几何对象上的点必须完全在栅格对象内。

示例

```
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)
```

5.7.6. ST_Update

用源raster更新目标raster。

语法

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

参数

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

示例

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

5.7.7. ST_MosaicFrom

将指定的raster对象进行镶嵌操作，合并成为一个新的raster对象。

语法

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

参数

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

描述

镶嵌函数会创建一个新的raster对象。

所有指定的raster对象需要满足以下条件：

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

涉及的数据库参数如下。

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

示例

```
INSERT INTO raster_table VALUES(1, ST_MosaicFrom(Array(SELECT raster_obj FROM raster_table WHERE id < 10), 'chunk_table_mosaic'))
UPDATE raster_table SET raster_obj = ST_MosaicFrom(Array(SELECT raster_obj FROM raster_table WHERE id < 10), 'chunk_table_mosaic') WHERE id = 11;
```

5.7.8. ST_MosaicTo

将raster对象或对象集镶嵌到目标raster对象。

语法

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

参数

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

描述

源raster对象和目标raster对象需要满足以下条件：

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

涉及的数据库参数如下。

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

示例

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

5.8. Overview操作

5.8.1. ST_BuildOverview

创建一个overview（概视图）。

语法

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

参数

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

函数支持创建一个基于表的raster overview对象。

所有涉及操作的raster对象需要满足以下条件：

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

示例

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

5.8.2. ST_UpdateOverview

使用raster对象或对象集更新overview。

语法

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

参数

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

描述

所有指定的raster对象需要满足以下条件：

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

示例

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

5.8.3. ST_EraseOverview

清空一个raster对象的指定区域。

语法

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

参数

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

示例

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

5.9. DEM操作

5.9.1. ST_Aspect

计算DEM坡向，返回坡向数组。

语法

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

参数

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

描述

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

坡向数据集中每个像元的值都可指示出该像元的坡度朝向。

示例

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

5.9.2. ST_Slope

计算坡度，返回弧度为单位的像元坡度栅格对象。

语法

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

参数

参数名称	描述
rast	raster对象。
pyramid_level	计算的金字塔等级。
Band	波段索引号
box	分析区域，格式为 '((m inX,m inY), (m axX,m axY))' 。

参数名称	描述
type	分析区域的坐标类型，只能是以下一种： • Raster（影像坐标） • World（世界坐标）
zfactor	高程夸张值，默认为1。
storageOption	目标raster对象的存储选项，参见ST_ClipToRast函数。

描述

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

示例

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

5.9.3. ST_Hillshade

计算山影，返回山影数组。

语法

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

参数

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

参数名称	描述
storageOption	目标raster对象的存储选项，参见ST_ClipToRast函数。

描述

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

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

示例

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

5.9.4. ST_Overflow

根据区域和区域中的降水量输出区域内代表水深度的像元值。

语法

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

参数

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

描述

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

示例

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

5.9.5. ST_Flow_direction

根据区域中的DEM高程值，计算区域中的水流方向。

语法

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

参数

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

描述

使用d8算法，根据区域中的DEM高程值，计算区域中的水流方向。

raster对象目前仅支持1波段的DEM数据。

示例

```
select st_flow_direction(rast, '((-202286.94,2232375.16),(-202135.0,2232225))'::box) from t_overflow where i
d =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)
```

5.10. 属性的查询与更新

5.10.1. ST_Name

获得raster对象的名称。如果没有定义名称，则返回空值。

语法

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

参数

参数名称	描述
rast	raster对象。

示例

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

5.10.2. ST_SetName

设置raster对象的名称。

语法

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

参数

参数名称	描述
rast	raster对象。
name	新的对象名称。

示例

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

5.10.3. ST_MetaData

获得raster对象的元数据，返回JSON格式。

语法

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

参数

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

示例

```
select ST_MetaData(raster_obj) from raster_table;
-- meta data with a name
SELECT ST_MetaData(raster_obj, 'swh#scale_factor')
FROM raster_table;
    st_metadata
----------------------
 0.0001488117874873806
-- all meta data
SELECT ST_MetaData(raster_obj, 'all')
FROM raster_table;
    st_metadata
----------------------
{"AREA_OR_POINT":"Area"}
-- meta data with a name of a band
SELECT ST_MetaData(raster_obj, 0, 'NETCDF_DIM_time')
FROM raster_table;
 st_metadata
-------------
 1043112
 -- all meta data of a band
SELECT ST_MetaData(raster_obj, 'all')
FROM raster_table;
                                    st_metadata
-------------------------------------------------------------------------------------------------------------------------------------------------------
{"add_offset":"4.907141431495487","long_name":"Significant height of combined wind waves and swell","
missing_value":"-32767","NETCDF_DI.
M_time":"1043112","NETCDF_VARNAME":"swh","scale_factor":"0.0001488117874873806","units":"m","_Fill
Value":"-32767"}
```

5.10.4. ST_Width

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

语法

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

参数

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

示例

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

5.10.5. ST_Height

获得raster对象的高度。

语法

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

参数

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

示例

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

5.10.6. ST_NumBands

获得raster对象的波段数。

语法

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

参数

参数名称	描述
raster_obj	raster对象。

示例

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

5.10.7. ST_Value

输出指定波段和行列号的像元值。

语法

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

参数

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

描述

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

示例

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

5.10.8. ST_RasterID

获得raster对象的UUID（通用唯一识别码）。

语法

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

参数

参数名称	描述
raster_obj	raster对象。

示例

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

5.10.9. ST_CellDepth

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

语法

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

参数

参数名称	描述
raster_obj	raster对象。

示例

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

5.10.10. ST_CellType

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

语法

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

参数

参数名称	描述
raster_obj	raster对象。

示例

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

5.10.11. ST_InterleavingType

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

语法

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

参数

参数名称	描述
raster_obj	raster对象。

示例

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

5.10.12. ST_TopPyramidLevel

获得raster对象的金字塔最高层级。

语法

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

参数

参数名称	描述
raster_obj	raster对象。

示例

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

5.10.13. ST_Extent

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

语法

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

参数

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

描述

csOption为坐标空间选项，取值如下：

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

示例

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

5.10.14. ST_ConvexHull

根据栅格的地理参考信息获得栅格对象的凸包。

语法

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

参数

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

描述

如下图所示，红色外框包含的部分为栅格对象的凸包。

示例

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

5.10.15. ST_Envelope

根据栅格的地理参考信息获得栅格对象的外包矩形。

语法

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

参数

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

描述

如下图所示，红色外框包含的部分为栅格对象的外包矩形。

示例

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

5.10.16. ST_Srid

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

语法

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

参数

参数名称	描述
raster_obj	raster对象。

示例

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

5.10.17. ST_SetSrid

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

语法

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

参数

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

描述

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

示例

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

5.10.18. ST_ScaleX

获得raster对象在空间参考系下X方向的像素宽度。

前提条件

raster对象必须已经进行空间参考。

语法

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

参数

参数名称	描述
raster_obj	raster对象。

示例

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

5.10.19. ST_ScaleY

获得raster对象在空间参考系下Y方向的像素宽度。

前提条件

raster对象必须已经进行空间参考。

语法

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

参数

参数名称	描述
raster_obj	raster对象。

示例

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

5.10.20. ST_SetScale

设置raster对象在空间参考系下X、Y方向的像素宽度。

语法

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

参数

参数名称	描述
raster_obj	raster对象。
scaleX	空间参考系下X方向像素宽。

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

示例

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

5.10.21. ST_SkewX

获得raster对象在X方向的旋转。

前提条件

raster对象必须已经进行空间参考。

语法

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

参数

参数名称	描述
raster_obj	raster对象。

示例

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

5.10.22. ST_SkewY

获得raster对象在Y方向的旋转。

前提条件

raster对象必须已经进行空间参考。

语法

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

参数

参数名称	描述
raster_obj	raster对象。

示例

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

5.10.23. ST_SetSkew

设置raster对象X、Y方向的旋转。

语法

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

参数

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

示例

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

5.10.24. ST_UpperLeftX

获得raster对象在空间参考系下左上角点X值。

前提条件

raster对象必须已经进行空间参考。

语法

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

参数

参数名称	描述
raster_obj	raster对象。

示例

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

5.10.25. ST_UpperLeftY

获得raster对象在空间参考系下左上角点Y值。

前提条件

raster对象必须已经进行空间参考。

语法

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

参数

参数名称	描述
raster_obj	raster对象。

示例

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

5.10.26. ST_SetUpperLeft

设置raster对象在空间参考系下左上角点X、Y值。

语法

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

参数

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

示例

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

5.10.27. ST_PixelWidth

获得raster对象在空间参考系下像素的宽度。

语法

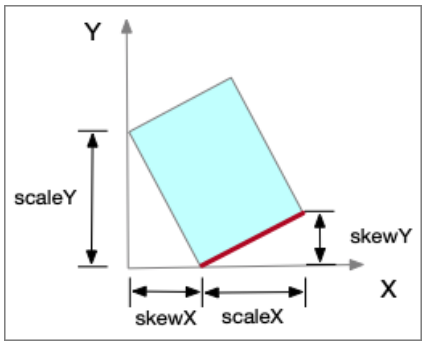
```
float8 ST_PixelWidth(raster raster_obj)
```

参数

参数名称	描述
raster_obj	raster对象。

描述

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



示例

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

5.10.28. ST_PixelHeight

获得raster对象在空间参考系下像素的高度。

语法

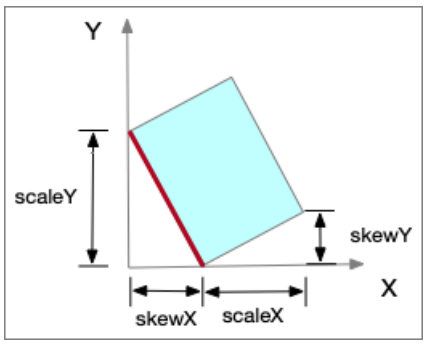
```
float8 ST_PixelHeight(raster raster_obj)
```

参数

参数名称	描述
raster_obj	raster对象。

描述

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



示例

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

5.10.29. ST_Georeference

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

语法

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

参数

参数名称	描述
raster_obj	raster对象。

示例

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

5.10.30. ST_IsGeoreferenced

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

语法

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

参数

参数名称	描述
raster_obj	raster对象。

描述

返回结果t表示true，f表示false。

示例

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

5.10.31. ST_UnGeoreference

去掉raster对象的地理参考信息。

语法

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

参数

参数名称	描述
rast	raster对象。

示例

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

5.10.32. ST_SetGeoreference

设置raster对象的地理参考信息。

语法

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

参数

参数名称	描述
rast	raster对象。

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

描述

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

示例

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

5.10.33. ST_RPCGeoreference

获取Raster对象的RPC（Rational Polynomial Coefficients）相关信息。如果Raster对象具备RPC信息，则返回一个JSON格式的字符串；如果不具备RPC信息，则返回NULL。

语法

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

参数

参数名称	描述
raster_obj	Raster对象。

描述

ST_RPCGeoreference函数返回的RPC参数是基于JSON格式的字符串，描述了RPC空间的参考信息。支持的参数如下：

参数名称	描述	类型
lineOff	线偏移	float
sampOff	采样偏移	float

参数名称	描述	类型
latOff	纬度偏移	float
longOff	经度偏移	float
heightOff	高程偏移	float
lineScale	线比例	float
sampScale	采样比例	float
latScale	纬度比例	float
longScale	经度比例	float
heightScale	高程比例	float
lineDenCoeff	线分母系数（20个）	float array
lineNumCoeff	线分子系数（20个）	float array
sampNumCoeff	采样分子系数（20个）	float array
sampDenCoeff	采样分母系数（20个）	float array
errBias	错误误差。以米为单位的图像中所有点的水平轴的均方根偏差误差，未知时为 `-1.0` 。	float
errRandom	随机误差。以米为单位的图像中每个点每水平轴的均方根随机误差，未知时为 `-1.0` 。	float

示例

```
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]}
```

5.10.34. ST_SetRPCGeoreference

设置Raster对象的RPC（Rational Polynomial Coefficients）参数信息。

语法

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

参数

参数名称	描述
raster_obj	Raster对象。
rpc_json	JSON格式的RPC描述信息。

描述

支持的RPC参数如下：

参数名称	描述	类型
lineOff	线偏移	float
sampOff	采样偏移	float
latOff	纬度偏移	float
longOff	经度偏移	float
heightOff	高程偏移	float
lineScale	线比例	float
sampScale	采样比例	float
latScale	纬度比例	float
longScale	经度比例	float
heightScale	高程比例	float
lineDenCoeff	线分母系数（20个）	float array
lineNumCoeff	线分子系数（20个）	float array
sampNumCoeff	采样分子系数（20个）	float array
sampDenCoeff	采样分母系数（20个）	float array
errBias	错误误差。以米为单位的图像中所有点的水平轴的均方根偏差误差，未知时为 `-1.0` 。	float
errRandom	随机误差。以米为单位的图像中每个点每水平轴的均方根随机误差，未知时为 `-1.0` 。	float

示例

```
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,
    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],"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]}
```

5.10.35. ST_NoData

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

语法

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

参数

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

示例

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

5.10.36. ST_SetNoData

设置raster对象的指定波段的NoData（无效值标识）的值。

语法

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

参数

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

示例

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

5.10.37. ST_ColorTable

获取raster对象的某一个波段的颜色表信息，返回颜色表的JSON格式。

语法

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

参数

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

描述

颜色表的JSON格式：

- 4分量：

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

- 3分量：

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

如果不存在颜色表，函数返回空值。

示例

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

5.10.38. ST_SetColorTable

设置raster对象的指定波段的颜色表信息，采用颜色表的JSON格式。

语法

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

参数

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

示例

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

5.10.39. ST_Statistics

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

语法

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

参数

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

示例

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

5.10.40. ST_SetStatistics

设置raster对象的指定波段的统计值信息。

语法

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

参数

参数名称	描述
rast	raster对象。
band	指定的波段序号，从0开始。
min,max,mean,std	统计值。
samplingParams	{"approx":false}或者 {"approx":true}。

示例

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

5.10.41. ST_SummaryStats

计算一个raster对象的指定波段集的统计值信息。

语法

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

参数

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

statsOptions用于指定统计参数，参数如下：

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

示例

计算raster对象的指定波段集的统计值信息。

```
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;
```

5.10.42. ST_ColorInterp

获取raster对象的某一个波段的颜色解释类型。

语法

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

参数

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

返回的interp值及其说明如下表。

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

示例

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

5.10.43. ST_SetColorInterp

设置raster对象的指定波段的颜色解释类型。

语法

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

参数

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

描述

interp枚举值及其解释：

值	说明
Undefined	颜色解释类型未定义。
GrayIndex	关联灰度颜色表。
RGBIndex	关联RGB颜色表。
RGBAIndex	关联RGBA颜色表。
CMYKIndex	关联CMYK颜色表。
HSLIndex	关联HSL颜色表。
RedBand	红色波段。
GreenBand	绿色波段。
BlueBand	蓝色波段。
AlphaBand	透明波段。
HueBand	HLS的色调分量。
SaturationBand	HLS的饱和度分量。
LightnessBand	HLS的亮度分量。
CyanBand	CMYK的青色波段。
MegentaBand	CMYK的品红波段。
YellowBand	CMYK的黄色波段。
BlackBand	CMYK的黑色波段。
YBand	YCBCR的亮度分量。

值	说明
CbBand	YCBCR的蓝色色度分量。
CrBand	YCBCR的红色色度分量。

示例

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

5.10.44. ST_Histogram

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

语法

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

参数

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

示例

```
select ST_Histogram(raster_obj, 0) from raster_table where id=1;
_____________________________________
{
 "approximate":false,
 "histsCounts":
 [2,1,1,0,8,17,47,101,193,345,443,640,877,1189,1560,1847,2087,2560,2816,3193,3567,3840,4101,4415,4498,387
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]
  }
 }
}
```

5.10.45. ST_SetHistogram

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

语法

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

参数

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

参数名称	描述
histogram	基于JSON描述的直方图。

描述

histogram基于JSON格式描述，具体的定义如下。

参数名称	描述	类型	说明
approximate	是否采用抽样方法	boolean	-
histsCounts	数据量数组	integer[]	-
binFunction/type	所采用的bin函数类型	string	• linear为线性模型 • logarithm为对数模型 • explicit为显式指定
binFunction/binTable/binValues	bin值	number[]	explicit有效
binFunction/binRange/minValue	最小值	number	logarithm linear有效
binFunction/binRange/m axValue	最大值	number	logarithm linear有效
binFunction/binRange/outRange	超出值	number	logarithm linear有效
binFunction/binRange/binValues	bin值	number[]	logarithm linear有效

```
eg.1:
{
 "approximate":false,
 "histsCounts":
 [2,1,1,0,8,17,47,101,193,345,443,640,877,1189,1560,1847,2087,2560,2816,3193,3567,3840,4101,4415,4498,38
76,3235,2458,1800,1598,1087,731,638,426,264,198,147,126,104,104,80,84,86,71,80,62,74,85,72,80,70,88,69,68,
62,58,63,51,53,55,54,56,55,63,47,39,49,59,66,62,64,73,66,72,67,84,86,79,91,92,117,138,136,142,157,225,287,28
5,382,449,567,628,750,855,1021,1142,1242,1410,1504,1590,1786,1870,2044,2099,2277,2373,2451,2585,2646,2
882,2878,3091,3396,3620,3911,4124,4304,4700,4893,5314,5446,5657,5765,5649,5749,5753,5601,5335,5161,49
43,4592,4445,4207,4083,4090,4270,4465,4514,4844,5204,5331,5597,5777,5838,6004,6316,6095,5762,5567,546
5,4923,4677,4220,3843,3401,3041,2571,2345,1972,1725,1376,1140,1008,841,716,548,442,373,308,212,133,78,6
8,31,24,12,2,2,1],
 "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
,134.0,135.0,136.0,137.0,138.0,139.0,140.0,141.0,142.0,143.0,144.0,145.0,146.0,147.0,148.0,149.0,150.0,151.0,1
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]
  }
 }
}
eg.2:
{
 "approximate" : true,
 "histsCounts" :[1,2,3,4,5],
 "binFunction" :
 {
  "type" : "explicit",
  "binTable" :
  {
   "binValues" : [1.0,2.0,3.0,4.0,5.0]
  }
 }
}
```

示例

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

5.10.46. ST_BuildHistogram

计算一个raster对象的指定波段集的直方图信息。

语法

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

参数

参数名称	描述
raster_obj	Raster对象。

示例

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

5.10.47. ST_StatsQuantile

计算raster对象的中位数。

语法

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

参数

参数名称	描述
raster_obj	raster对象。

描述

按照波段计算中位数，计算结果会记录到raster对象的元数据中。

示例

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

5.10.48. ST_Quantile

查询raster对象分位数的像素值。

前提条件

通过ST_StatsQuantile预先计算分位数。

语法

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

参数

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

示例

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

5.10.49. ST_MD5Sum

返回一个raster对象的MD5字符串。

语法

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

参数

参数名称	描述
raster_obj	raster对象。

描述

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

涉及的数据库参数如下。

参数	类型	说明
ganos.raster.calculate_md5	boolean	指定是否在raster对象入库时计算MD5并保存到元数据中。取值：true | false。默认值：false。示例： `Set ganos.raster.calculate_md5 = true;`
ganos.raster.md5sum_chunk_size	integer	指定计算MD5时每次读入的缓存大小。单位：MB。取值：1~256。默认值：10。示例： `Set ganos.raster.md5sum_chunk_size = 20;`

示例

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

5.10.50. ST_SetMD5Sum

设置raster对象的MD5字符串。

语法

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

参数

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

描述

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

涉及的数据库参数如下。

参数	类型	说明
ganos.raster.calculate_md5	boolean	指定是否在raster对象入库时计算MD5并保存到元数据中。取值：true | false。默认值：false。示例： `Set ganos.raster.calculate_md5 = true;`

示例

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

5.10.51. ST_XMin

获得raster对象在X方向的最小值。

语法

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

参数

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

描述

如果raster对象已经进行地理参考，返回的是X坐标的最小值，否则返回0。

示例

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

5.10.52. ST_YMin

获得raster对象在Y方向的最小值。

语法

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

参数

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

描述

如果raster对象已经进行地理参考，返回的是Y坐标的最小值，否则返回0。

示例

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

5.10.53. ST_XMax

获得raster对象在X方向的最大值。

语法

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

参数

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

描述

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

示例

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

5.10.54. ST_YMax

获得raster对象在Y方向的最大值。

语法

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

参数

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

描述

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

示例

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

5.10.55. ST_ChunkHeight

获得raster对象分块的高度。

语法

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

参数

参数名称	描述
raster_obj	raster对象。

描述

获得raster对象分块的高度，单位：像素。

示例

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

5.10.56. ST_ChunkWidth

获得raster对象分块的宽度。

语法

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

参数

参数名称	描述
raster_obj	raster对象。

描述

获得raster对象分块的宽度，单位：像素。

示例

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

5.10.57. ST_ChunkBands

获得raster对象分块波段维数量。

语法

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

参数

参数名称	描述
raster_obj	raster对象。

示例

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

5.10.58. ST_MetaItems

获得栅格对象自定义元数据项的名称列表。

语法

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

参数

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

示例

```
-- meta data items of a raster
SELECT ST_MetaItems(raster_obj)
FROM raster_table;
                                                st_metaitems
-------------------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------
 {latitude#long_name,latitude#units,longitude#long_name,longitude#units,NC_GLOBAL#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}
```

5.10.59. ST_SetMetaData

设置栅格对象或波段的元数据项。

语法

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

参数

参数名称	描述
raster_obj	需要计算的raster对象。

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

描述

如果传入的元数据值为空值（ '' ），会删除该元数据项。

示例

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

5.10.60. ST_BeginDateTime

获得栅格对象的开始时间。

语法

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

参数

参数名称	描述
raster_obj	需要计算的raster对象。

示例

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

5.10.61. ST_EndDateTime

获得栅格对象的结束时间。

语法

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

参数

参数名称	描述
raster_obj	需要计算的raster对象。

示例

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

5.10.62. ST_SetBeginDateTime

设置栅格对象的开始时间。

语法

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

参数

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

示例

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

5.10.63. ST_SetEndDateTime

设置栅格对象的结束时间。

语法

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

参数

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

示例

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

5.10.64. ST_DateTime

获得栅格对象或波段的时间。

语法

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

参数

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

描述

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

示例

```
SELECT ST_DateTime(raster_obj)
FROM raster_table;
                                                     st_datetime
------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------
{"0":"Mon Dec 31 00:00:00 2018","1":"Mon Dec 31 01:00:00 2018","2":"Mon Dec 31 02:00:00 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"
```

5.10.65. ST_SetDateTime

设置栅格对象的起始和终止时间，或波段时间。

语法

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

参数

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

示例

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

5.11. 代数与分析操作

5.11.1. ST_Reclassify

ST_Reclassify函数返回一个raster对象。 结果对象空间参考、分辨率与原始影像相同，波段数量通过relassexpr进行指定。

语法

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

参数

参数名称	描述
raster_obj	需要重分类的raster对象。
reclassexpr	JSON字符串用于表示分类数值区间。
storageOption	JSON字符串表示的返回结果的存储选项。

reclassexpr为JSON字符串数组，每个子JSON对象指明波段操作参数， 参数如下。

参数名称	描述	类型	默认值	说明
band	波段序号。	integer	无	波段序号，从0开始。
remap	分类采用的参数。	object	-	-
nodata	是否使用nodata。	boolean	false	• 如果为true，像素值为nodata，则分类结果也为nodata。 • 如果为false，则作为普通数值进行计算。
nodataValue	nodata值。	float8	0	新的nodata值。

remap表示如何将原始像素值映射到新像素值。

- 键表示原始像素值范围，可以由一个或多个数值构成，中间用英文逗号（,）进行分隔 。起始和结束可以指定开闭域关系。

- (表示大于
-) 表示小于
-] 表示小于等于
- [表示大于等于

默认为 `()` 。

- 值表示新旧像元值映射的结果，可以有一个或多个数值构成，中间用英文逗号（,）进行分隔 。
- 映射方式包含三种：
  - range --> range: 原始像素范围和新像素范围数值个数一致，； 例如"300,400,500":"80,90,100", "[300,400,500]":"80,90,100"。
  - range --> value: 原始像素范围比新像素范围数值个数多一个，例如"(300,400,500]":"80,90"。
  - value --> value: 原始像素范围和新像素范围数值都为一个，例如"10":"1"。
- 像素值不属于任何映射范围，则会被归纳到nodata。
- 同一个像素值不允许被多个范围包含。
- 示例
  - 示例1

    以下表示对波段0进行Reclassify操作：

    ```
    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"
       }
     }
    ]
    ```

  - 示例2

    支持多个分段值，不落入该区域的设置为nodata：

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

- 示例3

  以下表示对波段0进行Reclassify操作：

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

  以及对波段1的Reclassify操作：

```
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
 }
]
```

storageOption参数如下。

参数名称	描述	类型	默认值	说明
chunking	是否使用分块存储。	boolean	和原始raster一致	-
chunkdim	分块的维度信息。	string	和原始raster一致	在chunking=true时才有效。

参数名称	描述	类型	默认值	说明
chunktable	分块表名称。	string	''	如果传入''值，则会产生一个随机表名临时块表用于存放数据。 该临时表只在当前会话中有效。如果需要保持一个可访问的裁剪对象，则需要指定块表名称。
compression	压缩算法类型。	string	和原始raster一致	目前只支持none、jpeg、zlib、png、lzo和lz4。
quality	压缩质量。	integer	和原始raster一致	只针对jpeg压缩算法。
interleaving	交错方式。	string	和原始raster一致	必须是以下一种： • bip：Band interleaved by pixel • bil：Band nterleaved by pixel • bsq：Band Sequential
endian	字节序。	string	和原始raster一致	必须为以下其中之一： • NDR：Little endian • XDR：Big endian
celltype	像素类型。	string	和原始raster一致	-

示例

```
-- 永久表
CREATE TABLE rast_reclassify_result(id integer, rast raster);
-- 临时表
CREATE TEMP TABLE rast_reclassify_result_temp(id integer, rast raster);
-- 存放到临时表中
INSERT INTO rast_reclassify_result_temp(id, rast)
select 1, ST_Reclassify(rast, '[{"band":0,"remap":{"(0,100,200]":"20,50"}}]')
from reclass_table
```

5.11.2. ST_MapAlgebra

ST_MapAlgebra 函数通过使用代数计算表达式（Algebra Computing Language）对多个源对象的像素值进行计算，从而生成一个新的raster对象。

语法

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

参数

参数名称	描述
rasters	需要进行代数运算的raster对象数组。
algebraExpr	JSON字符串用于表示代数运算表达式。
storageOption	JSON字符串表示的返回结果的存储选项。

说明 传入的raster对象只要求长和宽一致，并不会对空间参考或分辨率等进行检查。如不一致可以通过ST_Tranform、ST_Resize和ST_Clip先进行处理。

algebraExpr为JSON字符串数组，每个子JSON对象指明代数运算表达式， 参数如下。

参数名称	描述	类型	默认值	说明
algebraExpr	代数运算表达式。	string	-	-
nodata	是否使用nodata。	boolean	false	• 如果为true，则如果像素值为nodata，栅格对象也为nodata。 • 如果为false，则作为普通数值进行计算。
nodataValue	nodata值。	float8	0	-

algebraExpr代数运算表达式有以下关键字。

- [r, b]
  - r：raster在数组中的id，0-n-1。
  - b：对应raster所在的波段号， 0-n-1。
- x

  该象元所在的列号。
- y

  该象元所在的行号。

表达式支持以下运算：

分类	运算符/函数	备注

分类	运算符/函数	备注
运算符	• + • - • * • / • % (remainder) • ** (power)	-
位运算	• << • >> • & • | • ^	-
逻辑运算	• < • > • == • != • <= • >= • && • || • !	-
运算函数	• abs • sqrt • exp • log • ln • sin • cos • tan • sinh • cosh • tanh • arcsin • arccos • arctan • ceil • floor • round	参数个数为1个。

分类	运算符/函数	备注
统计函数	• min • max • sum • mean • majority • minority • std • median • range • variety	参数个数至少为2个。

- 示例1

  本示例表示栅格对象为一个波段，结果值为raster[0]band[0] + raster[1]band[0] * raster[1]band[1]。

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

- 示例2

  本示例表示计算三个波段的方差。

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

- 示例3

  以下示例表生成三个图层，每个图层使用不同的表达式进行计算。

```
[
  {
    "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
  }
]
```

storageOption参数如下。

参数名称	描述	类型	默认值	说明
chunking	是否使用分块存储。	boolean	和原始raster一致	-
chunkdim	分块的维度信息。	string	和原始raster一致	在chunking=true时才有效。
chunktable	分块表名称。	string	''	如果传入''值，则会产生一个随机表名临时块表用于存放数据。 该临时表只在当前会话中有效。如果需要保持一个可访问的裁剪对象，则需要指定块表名称。
compression	压缩算法类型。	string	和原始raster一致	目前只支持none、jpeg、zlib、png、lzo和lz4。
quality	压缩质量。	integer	和原始raster一致	只针对jpeg压缩算法。
interleaving	交错方式。	string	和原始raster一致	必须是以下一种： • bip：Band interleaved by pixel • bil：Band nterleaved by pixel • bsq：Band Sequential
endian	字节序。	string	和原始raster一致	必须为以下其中之一： • NDR：Little endian • XDR：Big endian

示例

```
-- 永久表
CREATE TABLE rast_mapalgebra_result(id integer, rast raster);
-- 插入表中
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"}'
);
```

5.12. 栅格图像处理

5.12.1. ST_SubRaster

将影像的某个金字塔层级或某个波段作为一个新的raster进行返回。

语法

```
raster ST_SubRaster(raster raster_obj ,
            integer pyramidLevel default 0,
            cstring bands default '',
            cstring storageOption  default '',
            cstring options default '{}')
```

参数

参数名称	描述
raster_obj	raster对象。
pyramidLevel	金字塔层级。
bands	需要裁剪的波段。用 '0-2' 或者 ‘1,2,3’ 这种形式表示，从0开始。默认为 '' ，表示裁剪所有的波段。
storageOption	返回结果的存储选项，为JSON字符串。具体信息，请参见storageOption。
options	JSON字符串表示的操作选项。具体信息，请参见Options。

storageOption参数如下。

参数名称	描述	类型	默认值	说明
chunking	是否使用分块存储。	boolean	和原始raster一致。	无
chunkdim	分块的维度信息。	string	和原始raster一致。	当 `chunking=true` 时该参数才有效。
chunktable	分块表名称。	string	`''`	如果传入 `''` 或NULL，则会产生一个随机表名的临时块表用于存放数据。 该临时表只在当前会话中有效。如果需要保持一个可访问的裁剪对象，则需要指定块表名称。
compression	压缩算法类型。	string	和原始raster一致。	目前只支持如下类型： • NONE • JPEG • ZLIB • PNG • LZO • LZ4
quality	压缩质量。	integer	和原始raster一致。	取值范围为：1~99。 当**compression**参数为JPEG时该参数才有效。
interleaving	交错方式。	string	和原始raster一致。	必须是以下一种： • bip：Band interleaved by pixel • bil：Band nterleaved by pixel • bsq：Band Sequential
endian	字节序。	string	和原始raster一致。	必须是以下一种： • NDR：Little endian • XDR：Big endian

参数名称	描述	类型	默认值	说明
celltype	像素类型。	string	和原始raster一致。	必须为以下之一： • 1bb：1 bit • 2bui：2 bit unsigned integer • 4bui：4 bit unsigned integer • 8bsi：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 • 64bsi：64 bit signed integer • 64bui：64 bit unsigned integer • 32bf：32bit float • 64bf：64bit float

Options基于JSON格式的字符串，用于描述操作选项。支持的参数如下：

参数名称	描述	类型	默认值	说明
parallel	操作并行度。	integer	ganos.parallel.degree	并行度范围为1~64。
stretch	像素值拉伸方式。	string	none	支持以下几种方式： • none：不进行像素值拉伸，如果超出像素类型极值范围则用极值替代。 • stats：使用统计值进行拉伸。栅格对象必须要有统计值信息，可预先通过ST_SummaryStats函数进行计算。 • data_type：使用像素类型的极值进行像素值拉伸。

示例

将指定的波段转换为新的raster对象。

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

并行转换像素类型并进行拉伸操作。

```
SELECT ST_SubRaster(rast, 1, '0-2', '{"chunktable":"chunk_table", "chunking":true, "celltype": "8BUI"}', '{"st
retch": "data_type", "parallel": 4}')
FROM raster_sub
WHERE id=1;
```

5.12.2. ST_Transform

将源栅格对象进行投影变换，返回变换后的栅格对象。

语法

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

参数

参数名称	描述
rast	需要投影变换的raster对象。
outSrid	输出的影像的空间参考值，必须为有效的sird值（可以在表spatial_ref_sys查询到）。
processExpr	JSON字符串，指定重采样的方式以及nodata处理方式。
storageOption	返回结果的存储选项，为JSON字符串。

processExpr为JSON字符串数组，每个子JSON对象指定参数如下。

参数名称	描述	类型	默认值	说明
resample	重采样方式。	text	'Near'	栅格重采样方式，支持'Near'、'Average'、'Cubic'和'Bilinear'四种。
nodata	源影像的nodata值是否有效。	bool	false	• 如果为true，表示源影像的nodata是有效的，像元值为nodata的像元不参与重采样计算。 • 如果为false，表示源影像的nodata是无效的，像元值为nodata的像元参与重采样计算。

参数名称	描述	类型	默认值	说明
nodataValue	按波段指定新的nodata值。	float8 float8[]	NULL	nodataValue可指定为单个值或数组。 • 如果指定为单个值，表示输出栅格对象的所有波段使用同一个nodata值。 • 如果指定为数组，则数组元素个数必须与栅格的波段数一致。

说明 nodata与nodatavalue参数需谨慎使用，如果源栅格没有nodata，建议nodata设置为false，同时不需要指定nodatavalue，否则会出现结果影像失真的情况。

storageOption参数如下。

参数名称	描述	类型	默认值	说明
chunking	是否使用分块存储。	boolean	和原始raster一致	-
chunkdim	分块的维度信息。	string	和原始raster一致	在chunking=true时才有效。
chunktable	分块表名称。	string	''	如果传入''值，则会产生一个随机表名临时块表用于存放数据。 该临时表只在当前会话中有效。如果需要保持一个可访问的裁剪对象，则需要指定块表名称。
compression	压缩算法类型。	string	和原始raster一致	目前只支持none、jpeg、zlib、png、lzo和lz4。
quality	压缩质量。	integer	和原始raster一致	只针对jpeg压缩算法。
interleaving	交错方式。	string	和原始raster一致	必须是以下一种： • bip：Band interleaved by pixel • bil：Band nterleaved by pixel • bsq：Band Sequential
endian	字节序。	string	和原始raster一致	必须为以下其中之一： • NDR：Little endian • XDR：Big endian

说明 如果chunktable传入NULL或者''，则会在当前session中创建一个随机表名的临时表用于存放转换后的raster对象，该临时表只在当前会话中有效，会话结束临时表也随即删除。如果需要将转换的raster对象保存下来，则chunktable选项需要指定具体表名称。

示例

```
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', '{}'));
-----------------------------------------------------
-- 方式一：指定chunkTable名称，将转换结果持续化存储
-----------------------------------------------------
CREATE TABLE rat_transform_result(id integer, rast raster);
--不指定nodata
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;
--指定单个nodatavalue,同时nodata像元参与计算
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;
--指定nodata数组
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;
-----------------------------------------------------
-- 方式二：不指定chunktable名称，转换结果存放在随机表名的临时表中，仅可用于session内部的嵌套计算
-----------------------------------------------------
CREATE TEMP TABLE rat_transform_result_temp(id integer, rast raster);
INSERT INTO rat_transform_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;
```

5.12.3. ST_Rescale

对源栅格对象进行像素缩放，对应的地理空间范围保持不变，返回缩放后的栅格对象。

语法

```
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 '')
```

参数

参数名称	描述
rast	需要投影变换的raster对象。
scale_x	x方向的缩放比例。
scale_y	y方向的缩放比例。
scale_xy	x、y方向的缩放比例，两者相同。
processExpr	JSON字符串，指定重采样的方式以及nodata处理方式。
storageOption	返回结果的存储选项，为JSON字符串。

processExpr为JSON字符串数组，每个子JSON对象指定参数如下。

参数名称	描述	类型	默认值	说明
resample	重采样方式。	text	'Near'	栅格重采样方式，支持'Near'、'Average'、'Cubic'和'Bilinear'四种。
nodata	源影像的nodata值是否有效。	bool	false	• 如果为true，表示源影像的nodata是有效的，像元值为nodata的像元不参与重采样计算。 • 如果为false，表示源影像的nodata是无效的，像元值为nodata的像元参与重采样计算。
nodataValue	按波段指定新的nodata值。	float8 float8[]	NULL	nodataValue可指定为单个值或数组。 • 如果指定为单个值，表示输出栅格对象的所有波段使用同一个nodata值。 • 如果指定为数组，则数组元素个数必须与栅格的波段数一致。

> **说明** nodata与nodatavalue参数需谨慎使用，如果源栅格没有nodata，建议nodata设置为false，同时不需要指定nodatavalue，否则会出现结果影像失真的情况。

storageOption参数如下。

参数名称	描述	类型	默认值	说明
chunking	是否使用分块存储。	boolean	和原始raster一致	-
chunkdim	分块的维度信息。	string	和原始raster一致	在chunking=true时才有效。
chunktable	分块表名称。	string	''	如果传入''值，则会产生一个随机表名临时块表用于存放数据。 该临时表只在当前会话中有效。如果需要保持一个可访问的裁剪对象，则需要指定块表名称。
compression	压缩算法类型。	string	和原始raster一致	目前只支持none、jpeg、zlib、png、lzo和lz4。
quality	压缩质量。	integer	和原始raster一致	只针对jpeg压缩算法。
interleaving	交错方式。	string	和原始raster一致	必须是以下一种： • bip：Band interleaved by pixel • bil：Band nterleaved by pixel • bsq：Band Sequential
endian	字节序。	string	和原始raster一致	必须为以下其中之一： • NDR：Little endian • XDR：Big endian

> **说明** 如果chunktable传入NULL或者''，则会在当前session中创建一个随机表名的临时表用于存放转换后的raster对象，该临时表只在当前会话中有效，会话结束临时表也随即删除。如果需要将转换的raster对象保存下来，则chunktable选项需要指定具体表名称。

示例

```
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', '{}'));
-----------------------------------------------------
-- 方式一：指定chunkTable名称，将resize结果持续化存储
-----------------------------------------------------
CREATE TABLE rat_rescale_result(id integer, rast raster);
--不指定nodata
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;
--指定单个nodatavalue,同时nodata像元参与计算
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;
--指定nodata数组
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;
-----------------------------------------------------
-- 方式二：不指定chunktable名称，转换结果存放在随机表名的临时表中，仅可用于session内部的嵌套计算
-----------------------------------------------------
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;
```

5.12.4. ST_Resize

调整源栅格对象的像素范围，对应的地理空间范围保持不变，返回调整后的栅格对象。

语法

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

参数

参数名称	描述
rast	需要投影变换的raster对象。
outWidth	输出的栅格像素宽度。
outHeight	输出的栅格像素高度。
processExpr	JSON字符串，指定重采样的方式以及nodata处理方式。
storageOption	返回结果的存储选项，为JSON字符串。

processExpr为JSON字符串数组，每个子JSON对象指定参数如下。

参数名称	描述	类型	默认值	说明
resample	重采样方式。	text	'Near'	栅格重采样方式，支持'Near'、'Average'、'Cubic'和'Bilinear'四种。
nodata	源影像的nodata值是否有效。	bool	false	• 如果为true，表示源影像的nodata是有效的，像元值为nodata的像元不参与重采样计算。 • 如果为false，表示源影像的nodata是无效的，像元值为nodata的像元参与重采样计算。
nodataValue	按波段指定新的nodata值。	float8 float8[]	NULL	nodataValue可指定为单个值或数组。 • 如果指定为单个值，表示输出栅格对象的所有波段使用同一个nodata值。 • 如果指定为数组，则数组元素个数必须与栅格的波段数一致。

说明 nodata与nodatavalue参数需谨慎使用，如果源栅格没有nodata，建议nodata设置为false，同时不需要指定nodatavalue，否则会出现结果影像失真的情况。

storageOption参数如下。

参数名称	描述	类型	默认值	说明
chunking	是否使用分块存储。	boolean	和原始raster一致	-
chunkdim	分块的维度信息。	string	和原始raster一致	在chunking=true时才有效。

参数名称	描述	类型	默认值	说明
chunktable	分块表名称。	string	''	如果传入''值，则会产生一个随机表名临时块表用于存放数据。 该临时表只在当前会话中有效。如果需要保持一个可访问的裁剪对象，则需要指定块表名称。
compression	压缩算法类型。	string	和原始raster一致	目前只支持none、jpeg、zlib、png、lzo和lz4。
quality	压缩质量。	integer	和原始raster一致	只针对jpeg压缩算法。
interleaving	交错方式。	string	和原始raster一致	必须是以下一种： • bip：Band interleaved by pixel • bil：Band nterleaved by pixel • bsq：Band Sequential
endian	字节序。	string	和原始raster一致	必须为以下其中之一： • NDR：Little endian • XDR：Big endian

说明 如果chunktable传入NULL或者''，则会在当前session中创建一个随机表名的临时表用于存放转换后的raster对象，该临时表只在当前会话中有效，会话结束临时表也随即删除。如果需要将转换的raster对象保存下来，则chunktable选项需要指定具体表名称。

示例

```
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', '{}'));
-----------------------------------------------------
-- 方式一：指定chunkTable名称，将resize结果持续化存储
-----------------------------------------------------
CREATE TABLE rat_resize_result(id integer, rast raster);
--不指定nodata
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;
--指定单个nodatavalue,同时nodata像元参与计算
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;
--指定nodata数组
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;
-----------------------------------------------------
-- 方式二：不指定chunktable名称，转换结果存放在随机表名的临时表中，仅可用于session内部的嵌套计算
-----------------------------------------------------
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;
```

5.12.5. ST_RPCRectify

根据栅格影像的RPC（Rational Polynomial Coefficients）参数对栅格进行校正操作，返回校正后的栅格对象。

语法

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

参数

参数名称	描述
rast	需要校正的Raster对象。
dem	校正所参考的DEM栅格对象。如不需要请传入NULL。
rectifyOption	JSON字符串校正选项。
storageOption	返回结果的存储选项，格式为JSON字符串。
parallelOption	JSON字符串的并行选项。

描述

> **说明** 栅格对象必须具备RPC相关信息。

rectifyOption为JSON格式的字符串数组，每个子JSON对象指定参数如下：

参数名称	描述	类型	默认值	说明
resample	重采样方式	string	'Near'	栅格重采样方式，仅支持如下几种： • Near • Average • Cubic • Bilinear
outSrid	结果栅格空间参考	integer	与原Raster一致	无
nodataValue	按波段指定新的nodata值。	float8 float8[]	NULL	nodataValue可指定为单个值或数组。 • 如果指定为单个值表示输出栅格对象的所有波段使用同一个nodata值。 • 如果指定为数组[0, 0, 0]，则数组元素个数必须与栅格的波段数一致。 • 如不指定则优先使用原栅格数据nodata，如原栅格无nodata值，使用0作为nodata。
RPC_DEM_MISSING_VALUE	当DEM值为nodata或栅格数据在DEM范围外时使用的高程值。	float8	0	无

storageOption参数如下：

参数名称	描述	类型	默认值	说明
chunking	是否使用分块存储。	boolean	与原Raster一致	无
chunkdim	分块的维度信息。	string	与原Raster一致	只有chunking参数为true时，该参数有效。
chunktable	分块表名称。	string	''	如果传入 '' 值，则会在当前会话中创建一个随机表名的临时表用于存放转换后的Raster对象。该临时表只在当前会话中有效，会话结束后临时表也会随之删除。如果需要保存转换后的Raster对象，则需要指定具体表名。 如果您使用的并行方式不支持匿名表，则需要提前创建表。
compression	压缩算法类型。	string	与原Raster一致	目前仅支持如下类型： • NONE • JPEG • ZLIB • PNG • LZO • LZ4 • JP2K
quality	压缩质量。	integer	与原Raster一致	仅针对JPEG和压缩算法。
interleaving	交错方式。	string	与原Raster一致	仅支持如下几种方式： • bip：Band interleaved by pixel • bil：Band interleaved by line • bsq：Band Sequential
endian	字节序。	string	与原Raster一致	仅支持如下几种： • NDR：Little endian • XDR：Big endian

示例

```
CREATE TABLE if not exists raster_table(id integer, rast raster);
-- RPC栅格数据
INSERT INTO raster_table values(1, ST_ImportFrom('t_chunk','<RAST_DATA_DIR>/rpc.tif', '{}'));
-- DEM数据
INSERT INTO raster_table values(2, ST_ImportFrom('t_chunk','<RAST_DATA_DIR>/rpc_dem.tif', '{}'));
CREATE TABLE raster_result(id integer, rast raster);
--不指定dem，使用最邻近采样
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;
--指定dem，使用最邻近采样，输出srid为32635，并使用并行计算
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;
```

5.13. 操作符

5.13.1. 操作符（=）

判断两个raster的ID是否相同。

语法

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

参数

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

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

示例

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

5.13.2. 操作符（>）

判断两个raster的ID的比较关系。

语法

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

参数

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

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

示例

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

5.13.3. 操作符（<）

判断两个raster的ID的比较关系。

语法

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

参数

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

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

示例

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

5.13.4. 操作符（>=）

判断两个raster的ID的比较关系。

语法

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

参数

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

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

示例

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

5.13.5. 操作符（<=）

判断两个raster的ID的比较关系。

语法

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

参数

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

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

示例

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

5.14. 空间关系判断

5.14.1. ST_Intersects

判断raster和raster或raster和geometry的空间关系。

语法

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

参数

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

示例

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

5.14.2. ST_Contains

判断raster和raster或raster和geometry的空间关系。

语法

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

参数

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

示例

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

5.14.3. ST_ContainsProperly

判断raster和raster或raster和geometry的空间关系。

语法

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

参数

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

示例

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

5.14.4. ST_Covers

判断raster和raster或raster和geometry的空间关系。

语法

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

参数

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

示例

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

5.14.5. ST_CoveredBy

判断raster和raster或raster和geometry的空间关系。

语法

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

参数

参数名称	描述
rast1	raster对象1。

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

示例

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

5.14.6. ST_Disjoint

判断raster和raster或raster和geometry的空间关系。

语法

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

参数

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

示例

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

5.14.7. ST_overlaps

判断raster和raster或raster和geometry的空间关系。

语法

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

参数

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

示例

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

5.14.8. ST_Touches

判断raster和raster或raster和geometry的空间关系。

语法

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

参数

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

示例

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

5.14.9. ST_Within

判断raster和raster或raster和geometry的空间关系。

语法

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

参数

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

示例

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

5.15. 辅助函数

5.15.1. ST_CreateChunkTable

创建数据库块表。

语法

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

参数

参数名称	描述

参数名称	描述
tableName	数据库块表的名称。命名要求如下： • 以字母开头，以字母或数字结尾。 • 由小写字母、数字或下划线（_）组成。 • 长度为2~64个字符。 • 名称在当前数据库中必须是唯一的。

示例

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

5.15.2. ST_CheckGPU

验证是否有GPU环境。

语法

```
text  ST_CheckGPU();
```

描述

验证当前数据库运行环境是否有可识别的GPU硬件设备。

示例

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

5.15.3. ST_AKId

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

前提条件

raster对象必须存储在OSS。

语法

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

参数

参数名称	描述
raster_obj	raster对象。

示例

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

5.15.4. ST_SetAccessKey

设置raster对象OSS的登录信息。

前提条件

raster对象必须存储在OSS。

语法

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

参数

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

示例

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

5.15.5. ST_SetAKId

设置raster对象OSS的AccessKey ID。

前提条件

raster对象必须存储在OSS。

语法

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

参数

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

示例

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

5.15.6. ST_SetAKSecret

设置raster对象OSS的AccessKey Secret。

前提条件

raster对象必须存储在OSS。

语法

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

参数

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

示例

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

5.15.7. ST_RasterDrivers

获得所有Ganos Raster支持的数据源驱动列表。

语法

```
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);
```

参数

参数名称	描述
idx	驱动序号。
short_name	驱动名称缩写。
long_name	驱动名称全称。
can_read	返回值： • true：表示文件类型可以在ST_CreateRast和ST_ImportFrom函数中使用。 • false：表示文件类型在ST_CreateRast和ST_ImportFrom函数中使用会报错。
can_export	返回值： • true：表示文件类型可以在ST_ExportTo函数中使用。 • false：表示文件类型在ST_ExportTo函数中使用会报错。
can_asfile	返回值： • true：表示文件类型可以在ST_AsDatasetFile函数中使用。 • false：表示文件类型在ST_AsDatasetFile函数中使用会报错。
create_options	可以使用在ST_ExportTo和ST_AsDatasetFile时创建参数。

示例

```
-- 查询netCDF驱动相关信息。
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>
-- 查询所有 ST_ImportFrom和ST_CreateRast支持的驱动名称。
select short_name from
st_rasterdrivers()
where can_read =true;
-- 查询所有 ST_ExportTo 支持的驱动名称。
select short_name from
st_rasterdrivers()
where can_export =true;
-- 查询所有 ST_AsDatasetFile 支持的驱动名称。
select short_name from
st_rasterdrivers()
where can_asfile =true;
```

5.16. 变量

5.16.1. ganos.parallel.transaction

指定并行操作时并行事务是否可以和主事务一起提交或回滚。

数据类型

String

取值

- transaction_commit（默认值）：支持并行事务和主事务一起进行提交或回滚。
- fast_commit：不支持并行事务和主事务一起进行提交或回滚。

示例

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

5.16.2. ganos.parallel.degree

并行操作时，若未指定并行度，Ganos将会按照默认并行度执行并行操作。您可以通过ganos.parallel.degree变量来设置默认并行度。

数据类型

Integer

取值

取值范围：1~64。默认值为1。

示例

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

5.16.3. ganos.raster.calculate_md5

指定是否在导入raster对象时计算MD5值并记录到元数据中。

数据类型

Boolean

取值

- true：导入raster对象时会计算MD5值并记录到元数据中。
- false（默认值）：导入raster对象时不会计算MD5值。

示例

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

5.16.4. ganos.raster.md5sum_chunk_size

指定计算MD5时每次读取的缓存大小。

数据类型

Integer

取值

取值范围：10~100。单位：MB。默认值为10。

示例

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

5.16.5. ganos.raster.mosaic_must_same_nodata

指定镶嵌（mosaic）时数据源的nodata值是否必须相同。

数据类型

Boolean

取值

- true（默认值）：镶嵌时数据源的nodata值必须相同。
- false：镶嵌时数据源的nodata值可以不同。

> **说明** 镶嵌时并不会对nodata值进行转换，如果取值为false可能会导致镶嵌后的像素语义不一致。

示例

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

5.16.6. ganos.raster.memory_oss_file_max_size

指定ST_ImportFrom函数进行数据导入时，OSS文件映射到内存的最大值。

数据类型

integer

取值

取值范围：1 MB ~ 2048 MB。默认值100 MB。

示例

```
SET ganos.raster.memory_oss_file_max_size = 100;
```

6.SpatialRef SQL参考

6.1. ST_SrEqual

判断两个空间参考是否相同。

语法

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

参数

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

描述

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

示例

```
--比较两个基于文本的空间参考。
select ST_srEqual('GEOGCS["WGS 84",DATUM["WGS_1984",SPHEROID["WGS 84",6378137,298.257223563,AUTHORITY["EPSG","7030"]],AUTHORITY["EPSG","6326"]],PRIMEM["Greenwich",0,AUTHORITY["EPSG","8901"]],UNIT["degree",0.0174532925199433,AUTHORITY["EPSG","9122"]],AUTHORITY["EPSG","4326"]]', '+proj=longlat +datum=WGS84 +no_defs');
st_srequal
------------
 t
 --寻找spatial_ref_sys表中空间参考的srid。
 select srid from spatial_ref_sys where st_srequal(srtext::cstring, '+proj=longlat +ellps=GRS80 +no_defs ') limit 1;
 srid
------
 3824
```

6.2. ST_SrReg

注册一个新的空间参考。

语法

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

参数

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

描述

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

示例

```
--空间参考已存在
select 4490, ST_srReg('GEOGCS["China Geodetic Coordinate System 2000",DATUM["China_2000",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
--新空间参考
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
```

6.3. ST_SrFromEsriWkt

将一个Esri Wkt规范的字符串转为OGC规范的字符串。

语法

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

参数

参数名称	描述
sr1	基于Esri Wkt规范的空间参考字符串。

示例

```
SELECT ST_srFromEsriWkt('GEOGCS["China Geodetic Coordinate System 2000",DATUM["D_China_2000",SPHEROID["CGCS2000",6378137,298.257222101]],PRIMEM["Greenwich",0],UNIT["Degree",0.017453292519943295]]');
st_srfromesriwkt
-----------------------------------------------------------------------------------------------------------------------------------------------------------------
 GEOGCS["China Geodetic Coordinate System 2000",DATUM["China_2000",SPHEROID["CGCS2000",6378137,298.257222101]],PRIMEM["Greenwich",0],UNIT["Degree",0.017453292519943295]]
```

7.PointCloud SQL参考

7.1. 构造函数

7.1.1. ST_makePoint

构造一个pcpoint对象。

语法

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

参数

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

示例

```
SELECT ST_makePoint(1, ARRAY[-127, 45, 124.0, 4.0]);
-----------------------------------------------------
0101000000064CEFFFF94110000703000000400
```

7.1.2. ST_makePatch

构造一个pcpatch对象。

语法

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

参数

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

示例

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

7.1.3. ST_Patch

通过pcpoint数组构造一个pcpatch对象。

语法

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

参数

参数名称	描述
pts	pcpoint数组。

示例

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

7.2. 属性函数

7.2.1. ST_asText

将pcpoint/pcpatch对象转为JSON字符串表达。

语法

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

参数

参数名称	描述
pp	pcpatch对象。
pt	pcpoint对象。

示例

```
SELECT ST_asText('0101000000064CEFFFF94110000703000000400'::pcpoint);
-------------------------------
{"pcid":1,"pt":[-127,45,124,4]}
```

7.2.2. ST_pcID

获取pcpoint/pcpatch对象的schema ID。

语法

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

参数

参数名称	描述
pt	pcpoint对象。
pp	pcpatch对象。

示例

```
SELECT ST_pcID('0101000000064CEFFFF94110000703000000400'::pcpoint);
-------------------------------
1
```

7.2.3. ST_get

获取pcpoint对象属性值。

语法

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

参数

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

示例

```
SELECT ST_Get('0101000000064CEFFFF941100007030000000400'::pcpoint, 'Intensity');
------------------
4
SELECT ST_Get('0101000000064CEFFFF941100007030000000400'::pcpoint);
--------------------
{-127,45,124,4}
```

7.2.4. ST_numPoints

获取pcpatch对象中的pcpoint个数。

语法

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

参数

参数名称	描述
pc	pcpatch对象。

示例

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

7.2.5. ST_summary

获取pcpatch对象的概要信息。

语法

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

参数

参数名称	描述
pc	pcpatch对象。

示例

```
SELECT ST_Summary(pa) FROM patches LIMIT 1;
-----------------------------------
{"pcid":1, "npts":9, "srid":4326, "compr":"dimensional","dims":[{"pos":0,"name":"X","size":4,"type":"int32_
t","compr":"sigbits","stats":{"min":-126.99,"max":-126.91,"avg":-126.95}},{"pos":1,"name":"Y","size":4,"typ
e":"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}}]}
```

7.3. 对象操作

7.3.1. ST_compress

将pcpatch对象按指定方式压缩。

语法

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

参数

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

描述

压缩框架可以为：

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

当压缩框架为dimension时，压缩配置项可以为：

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

示例

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

7.3.2. ST_unCompress

将pcpatch对象解压。

语法

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

参数

参数名称	描述
pc	pcpatch对象。

描述

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

示例

```
SELECT ST_Uncompress(pa) FROM patches  WHERE ST_NumPoints(pa) = 1;
---------------------------------------
01010000000000000001000000C8CEFFFFF8110000102700000A00
```

7.3.3. ST_union

将pcpatch数组聚合成单个pcpatch对象。

语法

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

参数

参数名称	描述
pcs	pcpatch数组。

示例

```
-- Compare npoints(sum(patches)) to sum(npoints(patches))
SELECT ST_NumPoints(ST_Union(pa)) FROM patches;
SELECT Sum(ST_NumPoints(pa)) FROM patches;
100
```

7.3.4. ST_explode

将pcpatch对象反解为多行的pcpoint。

语法

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

参数

参数名称	描述
pc	pcpatch对象。

示例

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

7.3.5. ST_patchAvg

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

语法

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

参数

参数名称	描述
pc	pcpatch对象。

示例

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

7.3.6. ST_patchMax

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

语法

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

参数

参数名称	描述
pc	pcpatch对象。

示例

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

7.3.7. ST_patchMin

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

语法

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

参数

参数名称	描述
pc	pcpatch对象。

示例

```
SELECT ST_AsText(ST_PatchMin(pa)) FROM patches WHERE id = 7;
---------------------------------
{"pcid":1,"pt":[-126.5,45.5,50,5]}
```

7.3.8. ST_patchAvg

计算pcpatch对象中所有pcpoint某一属性维度的平均值。

语法

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

参数

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

示例

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

7.3.9. ST_patchMax

计算pcpatch对象中所有pcpoint某一属性维度的最大值。

语法

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

参数

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

示例

```
SELECT ST_PatchMax(pa, 'intensity') FROM patches WHERE id = 7;
--------------------------
125.0000000000000000
```

7.3.10. ST_patchMin

计算pcpatch对象中所有pcpoint某一属性维度的最小值。

语法

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

参数

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

示例

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

7.3.11. ST_filterGreaterThan

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

语法

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

参数

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

示例

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

7.3.12. ST_filterLessThan

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

语法

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

参数

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

示例

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

7.3.13. ST_filterEquals

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

语法

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

参数

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

示例

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

7.3.14. ST_filterBetween

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

语法

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

参数

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

描述

返回的结果中，不包括等于一大一小两个固定值的情况。

示例

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

7.3.15. ST_pointN

返回pcpatch中指定序号的pcpoint对象。

语法

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

参数

参数名称	描述
pc	pcpatch对象。

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

示例

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

7.3.16. ST_isSorted

判断pcpatch中所有pcpoint是否按指定的维度进行排序。

语法

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

参数

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

示例

```
SELECT ST_isSorted(pa, Array['a','c']) FROM patches;
-------------------------------------------------------------
f
(1 rows)
```

7.3.17. ST_sort

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

语法

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

参数

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

示例

```
update patches set pa=ST_Sort(pa, Array['y','x']);
---------------------------------------
(1 rows)
```

7.3.18. ST_range

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

语法

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

参数

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

示例

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

7.3.19. ST_setPcid

给pcpatch对象设置新的schema，返回新的pcpatch。

语法

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

参数

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

描述

在旧的schema中存在而新的schema中不存在的属性维将被丢弃。

示例

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

7.3.20. ST_transform

将pcpatch对象转换为新的schema，返回新的pcpatch。

语法

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

参数

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

描述

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

示例

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

7.3.21. ST_envelopeGeometry

返回pcpatch对象外包框geometry。

语法

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

参数

参数名称	描述
pc	pcpatch对象。

描述

外包框为ganos geometry的2D geometry对象。

示例

```
SELECT ST_AsText(ST_EnvelopeGeometry(pa)) FROM patches LIMIT 1;
------------------------------
POLYGON((-126.99 45.01,-126.99 45.09,-126.91 45.09,-126.91 45.01,-126.99 45.01))
CREATE INDEX ON patches USING GIST(ST_EnvelopeGeometry(patch));
```

7.3.22. ST_boundingDiagonalGeometry

返回pcpatch对象外包框对角线geometry。

语法

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

参数

参数名称	描述
pc	pcpatch对象。

描述

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

示例

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

7.4. OGC WKB操作

7.4.1. ST_asBinary

返回pcpoint对象的wkb值。

语法

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

参数

参数名称	描述
pt	pcpoint对象。

示例

```
SELECT ST_AsBinary('010100000064CEFFFF941100007030000000400'::pcpoint);
-----------------------------------------------
\x01010000800000000000c05fc0000000000080464000000000000005f40
```

7.4.2. ST_envelopeAsBinary

返回pcpatch对象外包框的wkb值。

语法

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

参数

参数名称	描述
pc	pcpatch对象。

描述

外包框为ganos geometry的2D geometry对象。

示例

```
SELECT ST_EnvelopeAsBinary(pa) FROM patches LIMIT 1;
------------------------------------------------
\x0103000000010000000500000090c2f5285cbf5fc0e17a
14ae4781464090c2f5285cbf5fc0ec51b81e858b46400ad7
a3703dba5fc0ec51b81e858b46400ad7a3703dba5fc0e17a
14ae4781464090c2f5285cbf5fc0e17a14ae47814640
```

7.4.3. ST_boundingDiagonalAsBinary

返回pcpatch对象外包框对角线的wkb值。

语法

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

参数

参数名称	描述
pc	pcpatch对象。

描述

外包框为2D geometry。

示例

```
SELECT ST_BoundingDiagonalAsBinary( ST_Patch(ARRAY[ ST_MakePoint(1, ARRAY[0.,0.,0.,10.]), ST_MakePoi
nt(1, ARRAY[1.,1.,1.,10.]), ST_MakePoint(1, ARRAY[10.,10.,10.,10.])]));
------------------------------------------------------------
\x01020000a0e61000000200000000000000000000000000000000000000000000000000000000000000000000000000000002440
0000000000002440000000000002440
```

7.5. 空间关系判断

7.5.1. ST_intersects

判断两个pcpatch对象的外包框是否相交。

语法

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

参数

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

示例

```
-- Patch should intersect itself
SELECT ST_Intersects(
    '0101000000000000000001000000C8CEFFFFF8110000102700000A00'::pcpatch,
    '0101000000000000000001000000C8CEFFFFF8110000102700000A00'::pcpatch);
------------------------
t
SELECT ST_Intersects('SRID=4326;POINT(-126.451 45.552)'::geometry, pa) FROM patches WHERE id = 7;
------------------------
t
```

7.6. 空间处理

7.6.1. ST_intersection

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

语法

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

参数

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

示例

```
SELECT ST_AsText(ST_Explode(ST_Intersection(
    pa,
    'SRID=4326;POLYGON((-126.451 45.552, -126.42 47.55, -126.40 45.552, -126.451 45.552))'::geometry
)))
FROM patches WHERE id = 7;
         st_astext
-------------------------------------
{"pcid":1,"pt":[-126.44,45.56,56,5]}
{"pcid":1,"pt":[-126.43,45.57,57,5]}
{"pcid":1,"pt":[-126.42,45.58,58,5]}
{"pcid":1,"pt":[-126.41,45.59,59,5]}
```

8.Trajectory SQL参考

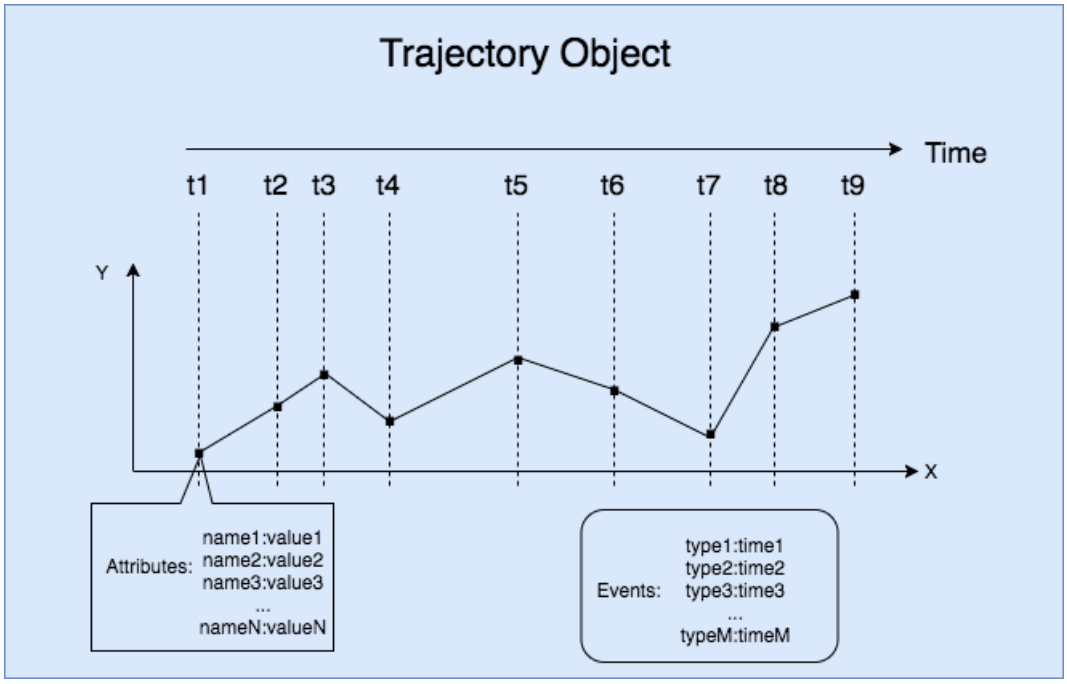
8.1. 基本概念

本章节将为您介绍Trajectory SQL的基本概念。

Trajectory对象

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

轨迹示意图如下。



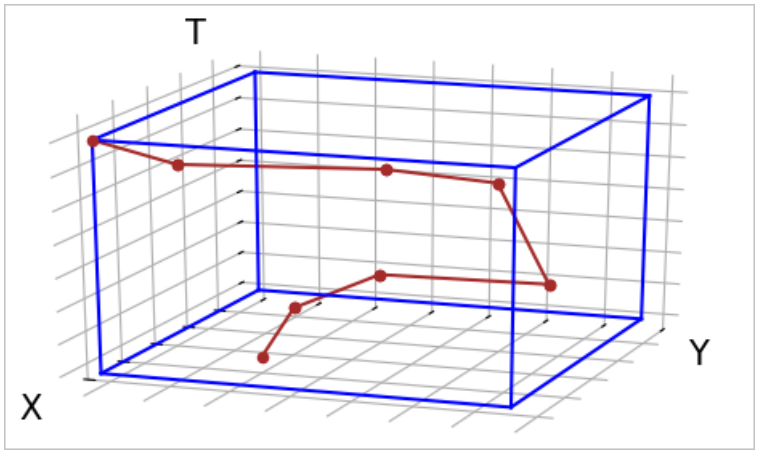
BoxNdf对象

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

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

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

外包矩形框示意图如下。



8.2. 构造函数

8.2.1. 构造函数概述

构造函数包括由JSON或数组构造轨迹对象的函数及轨迹追加函数。

8.2.2. ST_makeTrajectory

构造一个trajectory对象。

语法

```
trajectory  ST_makeTrajectory (leaftype type, geometry spatial, tsrange timespan , cstring attrs_json);
trajectory  ST_makeTrajectory (leaftype type, geometry spatial, timestamp start,  timestamp end ,  cstring a
ttrs_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[] a
ttr_field_names, int4[] attr_int4, float8[] attr_float8, text[] attr_cstring , anyarrayattr_any );
```

参数

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

1. attr_json的格式为：

```
{
 "leafcount": 3,
 "attributes": {
  "velocity": {
   "type": "integer",
   "length": 2,
   "nullable": true,
   "value": [
    120,
    null,
    140
   ]
```

```
  },
  "accuracy": {
   "type": "float",
   "length": 4,
   "nullable": false,
   "value": [
    120,
    130,
    140
   ]
  },
  "bearing": {
   "type": "float",
   "length": 8,
   "nullable": false,
   "value": [
    120,
    130,
    140
   ]
  },
  "vesname": {
   "type": "string",
   "length": 20,
   "nullable": true,
   "value": [
    "dsff",
    "fgsd",
    null
   ]
  },
  "active": {
   "type": "timestamp",
   "nullable": false,
   "value": [
    "Fri Jan 01 14:30:00 2010",
    "Fri Jan 01 15:00:00 2010",
    "Fri Jan 01 15:30:00 2010"
   ]
  }
 },
 "events": [
  {
   "1": "Fri Jan 01 14:30:00 2010"
  },
  {
   "2": "Fri Jan 01 15:00:00 2010"
  },
  {
   "3": "Fri Jan 01 15:30:00 2010"
  }
 ]
}
```

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

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

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

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

2. 如果传入的时间参数为 timespan 或者 start、end，则会根据spatial中点的个数进行插值。
3. 如果形式4不满足实际使用，可以自定义MakeTrajectory函数，前六个为固定参数，后面的参数可以根据实际情况进行定制：

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

示例

```
-- (1) ST_MakeTrajectory with timestamp range
select ST_MakeTrajectory('STPOINT'::leaftype, st_geomfromtext('LINESTRING (114 35, 115 36, 116 37)', 4326
), '[2010-01-01 14:30, 2010-01-01 15:30)'::tsrange, '{"leafcount":3,"attributes":{"velocity": {"type": "integer",
"length": 2,"nullable" : true,"value": [120, 130, 140]}, "accuracy": {"type": "float", "length": 4, "nullable" : fals
e,"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)
-- (2) ST_MakeTrajectory with start timestamp and end timestamp
select ST_MakeTrajectory('STPOINT'::leaftype, st_geomfromtext('LINESTRING (114 35, 115 36, 116 37)', 4326
), '2010-01-01 14:30'::timestamp, '2010-01-01 15:30'::timestamp, '{"leafcount":3,"attributes":{"velocity": {"ty
```

pe" : "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": [1
20, 130, 140]}, "vesname": {"type": "string", "length": 20, "nullable" : true,"value": ["adsf", "sdf", "sdfff"]}, "a
ctive": {"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 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) json is null
select ST_MakeTrajectory('STPOINT'::leaftype, st_geomfromtext('LINESTRING (114 35, 115 36, 116 37)', 4326
), '[2010-01-01 14:30, 2010-01-01 15:30)'::tsrange, null);
st_maketrajectory

{"trajectory":{"leafsize":3,"starttime":"Fri Jan 01 14:30:00 2010","endtime":"Fri Jan 01 15:30:00 2010","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 make 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)
```

8.2.3. ST_append

向轨迹中追加轨迹点或子轨迹。

语法

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

参数

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

示例

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

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

8.3. 编辑与处理函数

8.3.1. ST_Compress

将trajectory对象按一定规则进行压缩。

语法

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

参数

参数名称	描述
traj	原始轨迹。

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

描述

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

示例

```
--创建数据
Create table If not exists traj_test(id integer, mmsi integer,traj trajectory);
INSERT INTO traj_test(mmsi, traj) VALUES(477027500,ST_makeTrajectory('STPOINT'::leaftype, 'LINESTRING(
-179.48077 51.72814,-179.47416 51.73714,-179.47187 51.74027,-179.46964 51.74325,-179.46731 51.74634,-179.
46502 51.74934,-179.46183 51.75378,-179.45943 51.75736,-179.45560 51.76273,-179.44845 51.77186,-179.4341
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 09:59:09'::timestamp,'2017-01-15 10:00:20'::timestamp,'2017-01-15 10:01:29'::timestamp,'2017-01-15 10:02:39'::timestamp,'2017-01-15 10:06:29'::timestamp,'2017-01-15 10:07:40'::timestamp,'2017-01-15 10:15:00'::timestamp,'2017-01-15 10:16:20'::timestamp,'2017-01-15 10:17:29'::timestamp,'2017-01-15 11:30:09'::timestamp,'2017-01-15 11:33:58'::timestamp,'2017-01-15 11:36:58'::timestamp,'2017-01-15 11:42:00'::timestamp,'2017-01-15 11:43:10'::timestamp,'2017-01-15 11:44:20'::timestamp,'2017-01-15 11:50:28'::timestamp,'2017-01-15 18:01:00'::timestamp,'2017-01-15 18:54:13'::timestamp,'2017-01-15 18:55:21'::timestamp,'2017-01-15 18:56:22'::timestamp,'2017-01-15 18:58:21'::timestamp,'2017-01-15 19:13:21'::timestamp,'2017-01-15 19:15:21'::timestamp,'2017-01-15 19:16:41'::timestamp,'2017-01-15 19:20:11'::timestamp,'2017-01-15 19:22:31'::timestamp,'2017-01-15 19:23:41'::timestamp,'2017-01-15 19:26:10'::timestamp,'2017-01-15 20:15:49'::timestamp,'2017-01-15 20:34:39'::timestamp,'2017-01-15 20:35:49'::timestamp,'2017-01-15 20:37:30'::timestamp,'2017-01-15 20:39:00'::timestamp,'2017-01-15 20:40:19'::timestamp,'2017-01-15 20:41:30'::timestamp,'2017-01-15 20:43:08'::timestamp,'2017-01-15 20:44:19'::timestamp,'2017-01-15 20:46:19'::timestamp,'2017-01-15 20:47:29'::timestamp,'2017-01-15 20:48:40'::timestamp,'2017-01-15 20:49:49'::timestamp,'2017-01-15 20:50:59'::timestamp,'2017-01-15 20:52:21'::timestamp,'2017-01-15 20:53:29'::timestamp,'2017-01-15 20:54:50'::timestamp,'2017-01-15 20:57:09'::timestamp,'2017-01-15 20:58:20'::timestamp,'2017-01-15 20:59:40'::timestamp,'2017-01-15 21:00:49'::timestamp,'2017-01-15 21:01:50'::timestamp,'2017-01-15 21:02:58'::timestamp,'2017-01-15 21:04:40'::timestamp,'2017-01-15 21:05:50'::timestamp,'2017-01-15 21:07:29'::timestamp,'2017-01-15 21:09:11'::timestamp,'2017-01-15 21:12:49'::timestamp,'2017-01-15 21:14:30'::timestamp,'2017-01-15 21:18:30'::timestamp], '{"leafcount": 89,"attributes" : {"sog" : {"type":"float","length":8,"nullable":false,"value":[10.5,10.4,10.5,10.7,10.8,10.3,10.7,10.4,10.5,10.1,10.2,11.0,11.2,10.8,10.3,10.3,10.1,10.7,10.6,10.0,10.3,10.5,10.6,10.3,10.8,10.9,10.8,10.8,10.8,11.0,11.2,11.2,10.3,10.2,10.8,10.0,10.4,10.7,10.2,10.6,9.1,10.2,10.1,9.7,10.4,10.6,9.9,12.3,12.1,12.0,12.0,12.2,12.3,12.2,12.3,12.2,12.3,12.2,12.2,12.8,12.8,12.9,12.5,12.6,12.5,12.6,12.6,12.3,12.6,12.6,12.5,12.7,12.8,12.5,12.7,12.5,12.8,13.0,12.9,12.6,12.9,12.8,12.7,12.8,13.0,12.7,12.8,12.6,12.7]}, "cog" : {"type":"float","length":8,"nullable":false,"value":[23.3,25.7,25.9,23.6,25.3,24.1,23.0,21.6,20.7,24.8,22.4,28.5,23.1,30.3,26.2,28.1,25.1,28.7,31.4,28.2,30.4,29.4,29.2,23.0,25.1,25.1,23.5,22.7,27.1,23.3,19.2,27.1,31.0,28.8,22.0,30.1,24.6,26.2,26.7,24.7,26.8,29.5,19.9,30.1,28.8,28.7,30.0,74.2,69.1,75.2,81.3,81.3,80.1,72.6,82.4,74.2,74.9,67.7,73.4,74.2,72.2,80.5,78.6,77.3,70.9,80.1,85.4,71.9,67.0,77.5,77.5,72.2,70.5,72.6,70.8,77.8,71.2,71.2,73.8,75.4,67.1,77.5,74.3,76.9,80.1,72.8,76.0,75.4,72.9]},"heading" : {"type":"float","length":8,"nullable":false,"value":[22.0,23.0,23.0,23.0,23.0,21.0,21.0,25.0,24.0,26.0,25.0,27.0,28.0,29.0,31.0,30.0,28.0,29.0,29.0,28.0,28.0,27.0,24.0,24.0,25.0,25.0,25.0,26.0,25.0,24.0,25.0,29.0,31.0,28.0,29.0,31.0,28.0,29.0,29.0,29.0,27.0,27.0,26.0,26.0,26.0,27.0,27.0,69.0,71.0,72.0,72.0,71.0,73.0,72.0,72.0,72.0,72.0,71.0,71.0,72.0,72.0,72.0,73.0,72.0,71.0,72.0,72.0,72.0,71.0,72.0,72.0,73.0,71.0,72.0,71.0,72.0,73.0,72.0,72.0,72.0,72.0,73.0,74.0,73.0,73.0,73.0,73.0,73.0,73.0]}}}'));

--轨迹压缩

```
select st_compress(traj,0.001) as traj from traj_test;
```


```
                                                                traj
-------------------------------------------------------------------------------------------------------------------
 {"trajectory":{"version":1,"type":"STPOINT","leafcount":8,"start_time":"2017-01-15 09:06:39","end_time":"2017-01-15 21:18:30","spatial":"LINESTRING(-179.48077 51.72814,-179.42595 51.80094,-179.39734 51.83398,-179.37474 51.86568,-179.17504 52.11786,-179.14599 52.15132,-177.76666 52.85042,-176.68481 53.03327)","timeline":["2017-01-15 09:06:39","2017-01-15 09:34:40","2017-01-15 09:47:38","2017-01-15 09:59:09","2017-01-15 11:36:58","2017-01-15 11:50:28","2017-01-15 18:01:00","2017-01-15 21:18:30"],"attributes":{"leafcount":8,"sog":{"type":"float","length":8,"nullable":false,"value":[10.5,11.0,10.6,11.2,10.1,9.9,12.3,12.7]},"cog":{"type":"float","length":8,"nullable":false,"value":[23.3,28.5,29.2,27.1,19.9,30.0,74.2,72.9]},"heading":{"type":"float","length":8,"nullable":false,"value":[22.0,27.0,24.0,29.0,26.0,27.0,69.0,73.0]}}}}
(1 row)
```


```
select st_compress(traj,0.001,null,null) as traj from traj_test where traj_id=5;
```


```
                                                                traj
-------------------------------------------------------------------------------------------------------------------
{"trajectory":{"type":"STPOINT","leafsize":8,"starttime":"2017-01-15 09:06:39","endtime":"2017-01-15 21:18:30","spatial":"LINESTRING(-179.48077 51.72814,-179.42595 51.80094,-179.39734 51.83398,-179.37474 51.86568,-179.17504 52.11786,-179.14599 52.15132,-177.76666 52.85042,-176.68481 53.03327)","timeline":["2017-01-15 09:06:39","2017-01-15 09:34:40","2017-01-15 09:47:38","2017-01-15 09:59:09","2017-01-15 11:36:58","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)
```

8.3.2. ST_CompressSED

将trajectory对象按一定规则进行压缩。

语法

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

参数

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

描述

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

示例

```
select st_compressSED(traj, 0.001) as traj from traj_test;
        traj
-------------------------------------------------------------
{"trajectory":{"version":1,"type":"STPOINT","leafcount":12,"start_time":"2017-01-15 09:06:39","end_time":
"2017-01-15 21:18:30","spatial":"LINESTRING(-179.48077 51.72814,-179.42595 51.80094,-179.39734 51.83398,-
179.37474 51.86568,-179.18826 52.10105,-179.16482 52.12996,-179.14599 52.15132,-177.76666 52.85042,-177.
47319 52.90084,-177.31238 52.92697,-177.03751 52.97394,-176.68481 53.03327)","timeline":["2017-01-15 09:0
6:39","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,"nulla
ble":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","len
gth":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)
```

8.3.3. ST_attrDeduplicate

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

语法

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

参数

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

示例

```
select st_attrDeduplicate(ST_makeTrajectory('STPOINT'::leaftype, 'LINESTRING(-179.48077 51.72814,-179.4
6731 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 51.8
2505,-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','2017-01-1
5 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 0
9:48:49','2017-01-15 21:18:30'], '{"leafcount": 16, "attributes" : {"heading" : {"type": "float", "length": 4, "null
able" : 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]}}}' ),'headi
ng') as traj;
                              traj
---------------------------------------------------------------------------------------------------------------------------------------
--------------------------------
 {"trajectory":{"version":1,"type":"STPOINT","leafcount":6,"start_time":"2017-01-15 09:17:48","end_time":
"2017-01-15 21:18:30","spatial":"LINESTRING(-179.45943 51.75736,-179.44845 51.77
186,-179.40751 51.82223,-179.40497 51.82505,-179.39499 51.83709)","timeline":["2017-01-15 09:17:48","2017
-01-15 09:23:19","2017-01-15 09:38:09","2017-01-15 09:39:18","2017-01-15 21:18:3
0"],"attributes":{"leafcount":6,"heading":{"type":"float","length":4,"nullable":false,"value":[23.0,21.0,72.0,
73.0,74.0,73.0]}}}}
(1 row)
```

8.3.4. ST_sort

轨迹按时间序列从小到大重新排序，并返回新的轨迹对象。

语法

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

参数

参数名称	描述
traj	轨迹对象。

示例

```
select st_sort(ST_makeTrajectory('STPOINT'::leaftype, 'LINESTRING(-179.48077 51.72814,-179.46731 51.746
34,-179.46502 51.74934,-179.46183 51.75378,-179.45943 51.75736,-179.45560 51.76273,-179.44845 51.77186,-1
79.43419 51.78977,-179.41259 51.81643,-179.41001 51.81941,-179.40751 51.82223,-179.40497 51.82505,-179.4
0242 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 09:16:2
8'::timestamp,'2017-01-15 09:19:48'::timestamp,'2017-01-15 09:17:48'::timestamp,'2017-01-15 09:23:19'::time
stamp,'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,'2017-0
1-15 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":[0,1,2,3,4,5,6,7,8,9,10,1
1,12,13,14,15]}}}' ));
                                        st_sort
----------------------------------------------------------------------------------------------------------------------------------------------------------------
 {"trajectory":{"version":1,"type":"STPOINT","leafcount":16,"start_time":"2017-01-15 09:06:39","end_time"
:"2017-01-15 21:18:30","spatial":"LINESTRING(-179.48077 51.72814,-179.46502 51.7
4934,-179.46731 51.74634,-179.46183 51.75378,-179.4556 51.76273,-179.45943 51.75736,-179.44845 51.77186,-
179.41259 51.81643,-179.43419 51.78977,-179.41001 51.81941,-179.40751 51.82223,-
179.40497 51.82505,-179.40242 51.82796,-179.39981 51.83095,-179.39499 51.83709,-179.39734 51.83398)","ti
meline":["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","2
017-01-15 09:34:40","2017-01-15 09:36:59","2017-01-15 09:38:09","2017-01-15 09:
39:18","2017-01-15 09:40:40","2017-01-15 09:47:38","2017-01-15 09:48:49","2017-01-15 21:18:30"],"attributes
":{"leafcount":16,"heading":{"type":"integer","length":4,"nullable":false,"val
ue":[0,2,1,3,5,4,6,8,7,9,10,11,12,13,15,14]}}}}
(1 row)
```

8.3.5. ST_deviation

计算处理后的轨迹与原始轨迹之间的偏差。

语法

```
float8 ST_deviation(trajectory traj, trajectory after_oper_traj);
```

参数

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

示例

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

8.3.6. ST_removeDriftPoints

删除轨迹线中指定阈值的漂移点。

语法

```
trajectory ST_removeDriftPoints(trajectory traj, float8 speed, float8 dist, interval offset);
```

参数

参数名称	描述
traj	轨迹对象。
speed	速度阈值（m/s）。
dist	距离阈值（m）。
offset	时间间隔阈值。

描述

删除轨迹对象中满足条件的轨迹点，满足的条件要求轨迹点速度大于阈值speed（米/秒），且与上一轨迹点的距离大于阈值dist（米），时间间隔大于阈值offset，之后返回新的轨迹对象，该轨迹对象至少包含2个轨迹点。如果原始轨迹本身只有两个轨迹点，则不做任何删除。

参数speed是由轨迹位置和时间偏移值实时计算的结果，其单位固定为米/秒；距离阈值dist单位为米，时间阈值offset为时间间隔值。轨迹坐标空间参考系默认为WGS84，如果不为该空间参考系，请在构建轨迹时指定空间参考值或使用st_setsrid设置轨迹的空间参考值，最终会根据空间参考值统一计算球面距离（单位为米）。

示例

```
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;
--删除了2个漂移点
select id, st_leafcount(traj) from table_name where id = 1;
 id | st_leafcount
----+--------------
  1 |       49
(1 row)
```

8.3.7. ST_Split

用几何对象将一条轨迹切分为多条（子）轨迹。

语法

```
trajectory[] ST_Split(trajectory traj, geometry geom, float8 radius_of_buffer);
trajectory[] ST_Split(trajectory traj, text config);
```

参数

参数名称	描述
traj	被切分的轨迹对象。
geom	用于切分轨迹对象的空间几何对象，目前只支持point、multipoint两种类型。
radius_of_buffer	point构建buffer的半径（单位为米）。
config	切分轨迹的选项。

描述

- 选项一：`ST_Split(trajectory traj, geometry geom, float8 radius_of_buffer)`

  根据给定的几何对象切分轨迹对象，返回切分后的（子）轨迹数组；当用于切分轨迹对象的空间几何对象为multi类型时，表示轨迹有可能会被切分为多段。

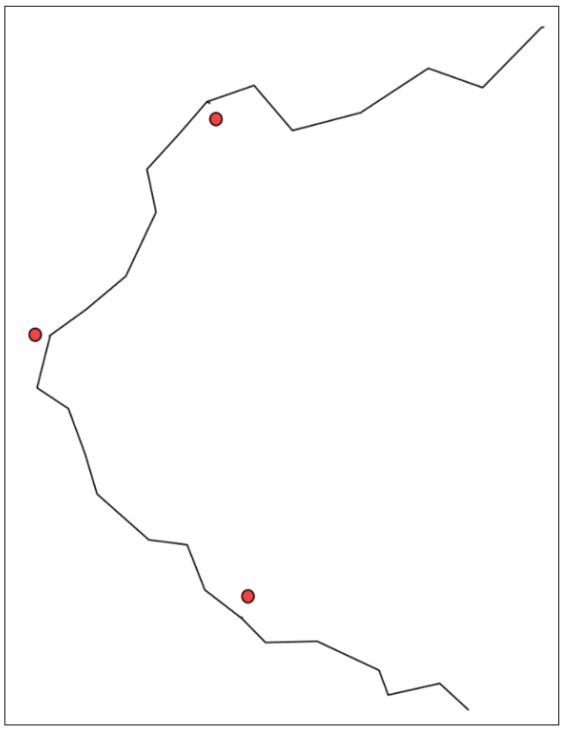
- 选项二：`ST_Split(trajectory traj, text config)`

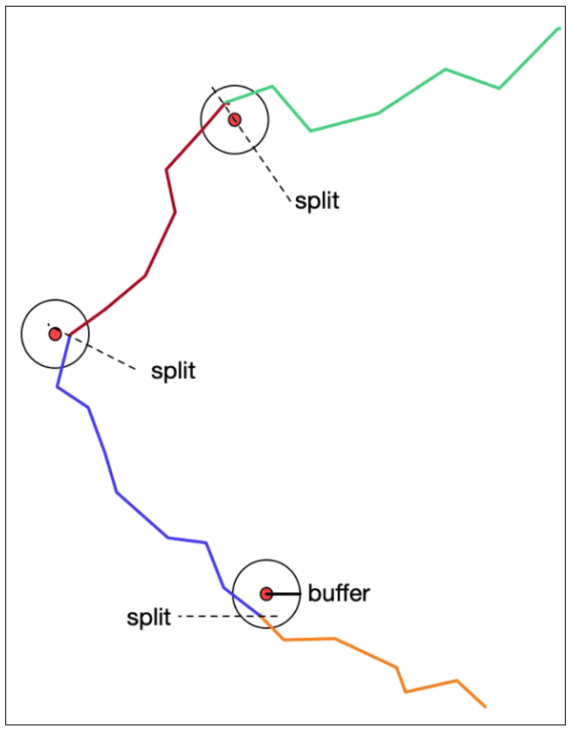
  根据提前设定的规则切分轨迹对象，返回切分后的（子）轨迹数组。目前包含以下可选规则，每次调用此函数时，仅可选择一个规则。

名称	类型	描述
cut_point.max_point	正整数	每间隔参数个点，将轨迹在采样点处切开成为两条子轨迹。
cut_point.even_divide	正整数	固定将每条轨迹切分为参数条子轨迹（如果轨迹所包含的边少于参数，则将每条边拆成一条子轨迹）。
cut_edge.time_interval	时间段字符串（同Interval类型），需为正。	每间隔固定时间间隔，将轨迹沿边切开成为两条子轨迹。
cut_edge.geohash	正偶数	按照参数位的GeoHash值（需确保源数据由经纬度表示）划分空间，切分轨迹使得每段子轨迹包含于一个GeoHash网格。
drop_edge.temporal_length	时间段字符串（同Interval类型）。	将时间间隔超过参数的边删除。
drop_edge.spatial_distance_2d	浮点数	将空间跨度超过参数的边删除（按照二维欧氏距离计算）。

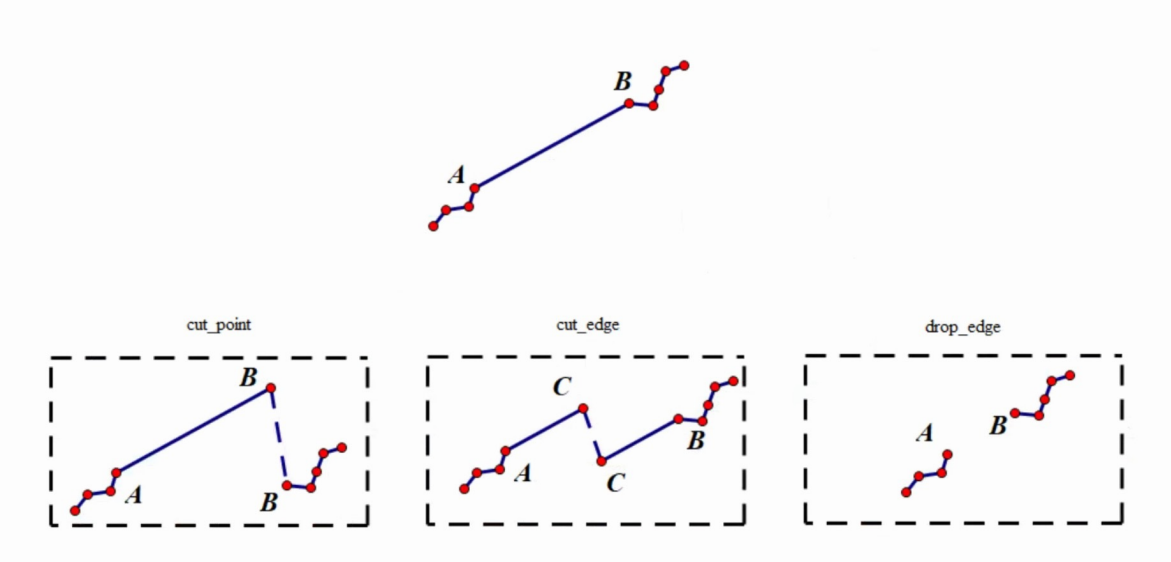
示意图

- 按几何对象切分。





- 按规则切分：cut_point寻找切分点（如下图所示，选择了B点切开），cut_edge在合适的位置切开线（如下图所示，在线段AB上选择了点C，在点C处切开），drop_edge删除两点间的线段（如下图所示，线段AB被删除）。



示例

```
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
```

```
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 -64.5007972046733,-46.9800617334743 -49.4026757289345 -61.6160059720278,-41.7122942996211 -46.3224360072054 -56.5283147455193)","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 -44.1693710885011,-23.1585765127093 -26.5895827477798 -40.6539742602035,-16.7020264320696 -21.6133877349397 -37.3055470525287)","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 -24.3842111621052,0.488159407706304 -3.68223926316326 -19.9478872027248,6.33406881305078 4.54123636645575 -15.0410129944794)","timeline":["2000-01-01 02:16:44","2000-01-01 02:28:01","2000-01-01 02:39:18","2000-01-01 02:50:35","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 -15.0410129944794,15.6666049417108 10.5611746329814 -11.2770220567472,14 11 -10)","timeline":["2000-01-01 03:01:52","2000-01-01 03:13:09","2000-01-01 03:24:26"]}}
(5 rows)

8.4. 空间参考系处理函数

8.4.1. ST_SRID

获得轨迹的空间参考标识符（SRID）。执行空间相关操作时，将使用SRID查找空间参考系信息，确保正确执行计算。

语法

```
int ST_SRID(trajectory traj);
```

参数

参数名称	描述
traj	轨迹对象。

描述

- SRID对应的地理坐标系统和投影信息保存在系统表spatial_ref_sys中。
- SRID默认值为0。

示例

```
select ST_SRID('{"trajectory":{"version":1,"type":"STPOINT","leafcount":2,"start_time":"2007-05-27 00:00:0
0","end_time":"2007-05-28 08:36:47.846","spatial":"LINESTRING(13.43593 52.41721,13.43593 52.41721)","ti
meline":["2007-05-27 00:00:00","2007-05-28 08:36:47.846"]}}'::trajectory);
 st_srid
---------
     0
(1 row)
```

8.4.2. ST_SetSRID

更改某条轨迹的空间参考标识符（SRID）。

语法

```
trajectory ST_SetSRID(trajectory traj, int srid);
```

参数

参数名称	描述
traj	轨迹对象。
srid	目标SRID。

描述

将轨迹的SRID设置为给定的值，但不更改其它内容。

示例

```
select ST_setSRID('{"trajectory":{"version":1,"type":"STPOINT","leafcount":2,"start_time":"2007-05-27 00:0
0:00","end_time":"2007-05-28 08:36:47.846","spatial":"SRID=4326;LINESTRING(13.43593 52.41721,13.43593 5
2.41721)","timeline":["2007-05-27 00:00:00","2007-05-28 08:36:47.846"]}}'::trajectory
,4002);
                                                    st_setsrid

----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------
 {"trajectory":{"version":1,"type":"STPOINT","leafcount":2,"start_time":"2007-05-27 00:00:00","end_time":
"2007-05-28 08:36:48","spatial":"SRID=4002;LINESTRING(13.43593 52.41721,13.43593 52.41721)","timeline":[
"2007-05-27 00:00:00","2007-05-28 08:36:48"]}}
(1 row)
```

8.4.3. ST_Transform

将轨迹从一个空间参考系转换到另一个空间参考系。

语法

```
trajectory ST_Transform(trajectory traj, int srid);
```

参数

参数名称	描述
traj	需要转换的轨迹对象。
srid	目标空间参考系的SRID。

> **说明** 原轨迹对象必须有SRID属性才能被转换到另一个空间参考系。

示例

```
select ST_Transform('{"trajectory":{"version":1,"type":"STPOINT","leafcount":4,"start_time":"2020-11-03 0
8:00:00","end_time":"2020-11-03 18:03:20","spatial":"SRID=4326;LINESTRING(114.49211 37.97999,114.49211
37.97521,114.49191 37.98021,124 37)","timeline":["2020-11-03 08:00:00","2020-11-03 08:01:40","2020-11-03 0
8:03:20","2020-11-03 18:03:20"]}}'::trajectory, 2401);
                                                                                                st_transform

----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------
 {"trajectory":{"version":1,"type":"STPOINT","leafcount":4,"start_time":"2020-11-03 08:00:00","end_time":
"2020-11-03 18:03:20","spatial":"SRID=2401;LINESTRING(29019418.8397 5025539.47515,29019696.5878 5024
992.53808,29019387.9428 5025555.44863,29954166.377 5430882.06455)","timeline":["2020-11-03 08:00:00","
2020-11-03 08:01:40","2020-11-03 08:03:20","2020-11-03 18:03:20"]}}
(1 row)
```

8.5. 属性元数据

8.5.1. ST_attrDefinition

获得轨迹属性定义。

语法

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

参数

参数名称	描述
traj	需要获得属性定义轨迹。

示例

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

8.5.2. ST_attrSize

获得轨迹的属性数量。

语法

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

参数

参数名称	描述
traj	轨迹对象。

示例

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

8.5.3. ST_attrName

获得轨迹的属性名称。

语法

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

参数

参数名称	描述
traj	轨迹对象。
index	属性索引号，从0开始。

示例

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

8.5.4. ST_attrType

获得轨迹属性类型。

语法

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

参数

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

描述

返回描述属性类型的字符串为以下几种之一：

- integer
- float
- string
- timestamp
- bool

示例

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

8.5.5. ST_attrLength

获得轨迹属性定义长度。

语法

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

参数

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

示例

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

8.5.6. ST_attrNullable

获得轨迹属性是否允许为空。

语法

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

参数

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

示例

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

8.6. 事件函数

8.6.1. ST_addEvent

给轨迹增加一个事件。

语法

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

参数

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

描述

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

示例

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

st_addevent

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

8.6.2. ST_eventTimes

获得轨迹的所有事件时间。

语法

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

参数

参数名称	描述
traj	轨迹对象。

示例

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

8.6.3. ST_eventTime

获得轨迹指定索引事件时间。

语法

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

参数

参数名称	描述
traj	轨迹对象。
index	事件索引序号。

示例

```
With traj AS (
 select '{"trajectory":{"version":1,"type":"STPOINT","leafcount":2,"start_time":"2010-01-01 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,null]},"speed":{"type":"float","length":8,"nullable":true,"value":[null,1.0]},"angel":{"type":"string","
length":64,"nullable":true,"value":["test",null]}, "tngel2":{"type":"timestamp","length":8,"nullable":true,"v
alue":["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)
```

8.6.4. ST_eventTypes

获得轨迹的所有事件类型。

语法

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

参数

参数名称	描述
traj	轨迹对象。

示例

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

8.6.5. ST_eventType

获得轨迹指定索引事件的类型。

语法

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

参数

参数名称	描述
traj	轨迹对象。
index	事件序号。

示例

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

8.7. 属性函数

8.7.1. ST_startTime

获得轨迹的起始时间。

语法

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

参数

参数名称	描述
traj	需要获得开始时间的轨迹。

示例

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

8.7.2. ST_endTime

获得轨迹的结束时间。

语法

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

参数

参数名称	描述
traj	需要获得结束时间的轨迹。

示例

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

8.7.3. ST_trajectorySpatial

获得轨迹的几何对象。

语法

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

参数

参数名称	描述
traj	需要获得几何对象的轨迹。

示例

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

8.7.4. ST_trajectoryTemporal

获得轨迹的时间线。

语法

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

参数

参数名称	描述
traj	需要获得时间线的轨迹。

示例

```
select ST_trajectoryTemporal(ST_MakeTrajectory('STPOINT'::leaftype, st_geomfromtext('LINESTRING (114
35, 115 36, 116 37)', 4326), '[2010-01-01 14:30, 2010-01-01 15:30)'::tsrange, null));
                    st_trajectorytemporal
-------------------------------------------------------------------------------
{"timeline":["2010-01-01 14:30:00","2010-01-01 15:00:00","2010-01-01 15:30:00"]}
(1 row)
```

8.7.5. ST_trajAttrs

获得轨迹的属性信息。

语法

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

参数

参数名称	描述
traj	需要获得属性信息的轨迹。

示例

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

8.7.6. ST_attrIntMax

获得轨迹属性字段类型为integer的属性最大值。

语法

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

参数

参数名称	描述
traj	需要获得属性信息的轨迹。

参数名称	描述
attr_field_name	指定的属性字段名称。

示例

```
select st_attrIntMax(ST_makeTrajectory('STPOINT'::leaftype, 'LINESTRING(-179.48077 51.72814,-179.46731 51.74634,-179.46502 51.74934,-179.46183 51.75378,-179.45943 51.75736,-179.45560 51.76273,-179.44845 51.77186,-179.43419 51.78977,-179.41259 51.81643,-179.41001 51.81941,-179.40751 51.82223,-179.40497 51.82505,-179.40242 51.82796,-179.39981 51.83095,-179.39734 51.83398,-179.39499 51.83709)'::geometry, ARRAY['2017-01-15 09:06:39'::timestamp,'2017-01-15 09:14:48','2017-01-15 09:13:39','2017-01-15 09:16:28','2017-01-15 09:19:48','2017-01-15 09:17:48','2017-01-15 09:23:19','2017-01-15 09:34:40','2017-01-15 09:30:28','2017-01-15 09:36:59','2017-01-15 09:38:09','2017-01-15 09:39:18','2017-01-15 09:40:40','2017-01-15 09:47:38','2017-01-15 21:18:30','2017-01-15 09:48:49'], '{"leafcount": 16, "attributes" : {"heading" : {"type": "integer", "length": 4, "nullable" : false,"value":[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]}}}'), 'heading');
 st_attrintmax
---------------
        15
```

8.7.7. ST_attrIntMin

获得轨迹属性字段类型为integer的属性最小值。

语法

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

参数

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

示例

```
select st_attrIntMin(ST_makeTrajectory('STPOINT'::leaftype, 'LINESTRING(-179.48077 51.72814,-179.46731 51.74634,-179.46502 51.74934,-179.46183 51.75378,-179.45943 51.75736,-179.45560 51.76273,-179.44845 51.77186,-179.43419 51.78977,-179.41259 51.81643,-179.41001 51.81941,-179.40751 51.82223,-179.40497 51.82505,-179.40242 51.82796,-179.39981 51.83095,-179.39734 51.83398,-179.39499 51.83709)'::geometry, ARRAY['2017-01-15 09:06:39'::timestamp,'2017-01-15 09:14:48','2017-01-15 09:13:39','2017-01-15 09:16:28','2017-01-15 09:19:48','2017-01-15 09:17:48','2017-01-15 09:23:19','2017-01-15 09:34:40','2017-01-15 09:30:28','2017-01-15 09:36:59','2017-01-15 09:38:09','2017-01-15 09:39:18','2017-01-15 09:40:40','2017-01-15 09:47:38','2017-01-15 21:18:30','2017-01-15 09:48:49'], '{"leafcount": 16, "attributes" : {"heading" : {"type": "integer", "length": 4, "nullable" : false,"value":[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]}}}'), 'heading');
 st_attrintmin
---------------
        0
```

8.7.8. ST_attrIntAverage

获得轨迹属性字段类型为integer的属性平均值。

语法

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

参数

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

示例

```
select st_attrIntAverage(ST_makeTrajectory('STPOINT'::leaftype, 'LINESTRING(-179.48077 51.72814,-179.46731 51.74634,-179.46502 51.74934,-179.46183 51.75378,-179.45943 51.75736,-179.45560 51.76273,-179.44845 51.77186,-179.43419 51.78977,-179.41259 51.81643,-179.41001 51.81941,-179.40751 51.82223,-179.40497 51.82505,-179.40242 51.82796,-179.39981 51.83095,-179.39734 51.83398,-179.39499 51.83709)'::geometry, ARRAY['2017-01-15 09:06:39'::timestamp,'2017-01-15 09:14:48','2017-01-15 09:13:39','2017-01-15 09:16:28','2017-01-15 09:19:48','2017-01-15 09:17:48','2017-01-15 09:23:19','2017-01-15 09:34:40','2017-01-15 09:30:28','2017-01-15 09:36:59','2017-01-15 09:38:09','2017-01-15 09:39:18','2017-01-15 09:40:40','2017-01-15 09:47:38','2017-01-15 21:18:30','2017-01-15 09:48:49'], '{"leafcount": 16, "attributes" : {"heading" : {"type": "integer", "length": 4, "nullable" : false,"value":[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]}}}'), 'heading');
 st_attrintaverage
-------------------
                 7
```

8.7.9. ST_attrFloatMax

获得轨迹属性字段类型为float的属性最大值。

语法

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

参数

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

示例

```
select st_attrFloatMax(ST_makeTrajectory('STPOINT'::leaftype, 'LINESTRING(-179.48077 51.72814,-179.46731 51.74634,-179.46502 51.74934,-179.46183 51.75378,-179.45943 51.75736,-179.45560 51.76273,-179.44845 51.77186,-179.43419 51.78977,-179.41259 51.81643,-179.41001 51.81941,-179.40751 51.82223,-179.40497 51.82505,-179.40242 51.82796,-179.39981 51.83095,-179.39734 51.83398,-179.39499 51.83709)'::geometry, ARRAY['2017-01-15 09:06:39'::timestamp,'2017-01-15 09:13:39','2017-01-15 09:14:48','2017-01-15 09:16:28','2017-01-15 09:17:48','2017-01-15 09:19:48','2017-01-15 09:23:19','2017-01-15 09:30:28','2017-01-15 09:34:40','2017-01-15 09:36:59','2017-01-15 09:38:09','2017-01-15 09:39:18','2017-01-15 09:40:40','2017-01-15 09:47:38','2017-01-15 09:48:49','2017-01-15 21:18:30'], '{"leafcount": 16, "attributes" : {"heading" : {"type": "float", "length": 4, "nullable" : false,"value": [23.0,23.0,23.0,23.0,21.0,21.0,72.0,72.0,72.0,72.0,73.0,74.0,73.0,73.0,73.0,73.0]}}}' ),'heading');
 st_attrfloatmax
-----------------
         74
```

8.7.10. ST_attrFloatMin

获得轨迹属性字段类型为float的属性最小值。

语法

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

参数

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

示例

```
select st_attrFloatMin(ST_makeTrajectory('STPOINT'::leaftype, 'LINESTRING(-179.48077 51.72814,-179.46731 51.74634,-179.46502 51.74934,-179.46183 51.75378,-179.45943 51.75736,-179.45560 51.76273,-179.44845 51.77186,-179.43419 51.78977,-179.41259 51.81643,-179.41001 51.81941,-179.40751 51.82223,-179.40497 51.82505,-179.40242 51.82796,-179.39981 51.83095,-179.39734 51.83398,-179.39499 51.83709)'::geometry, ARRAY['2017-01-15 09:06:39'::timestamp,'2017-01-15 09:13:39','2017-01-15 09:14:48','2017-01-15 09:16:28','2017-01-15 09:17:48','2017-01-15 09:19:48','2017-01-15 09:23:19','2017-01-15 09:30:28','2017-01-15 09:34:40','2017-01-15 09:36:59','2017-01-15 09:38:09','2017-01-15 09:39:18','2017-01-15 09:40:40','2017-01-15 09:47:38','2017-01-15 09:48:49','2017-01-15 21:18:30'], '{"leafcount": 16, "attributes" : {"heading" : {"type": "float", "length": 4, "nullable" : false,"value": [23.0,23.0,23.0,23.0,21.0,21.0,72.0,72.0,72.0,72.0,73.0,74.0,73.0,73.0,73.0,73.0]}}}' ),'heading');
 st_attrfloatmin
-----------------
         21
```

8.7.11. ST_attrFloatAverage

获得轨迹属性字段类型为float的属性平均值。

语法

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

参数

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

示例

```
select st_attrFloatAverage(ST_makeTrajectory('STPOINT'::leaftype, 'LINESTRING(-179.48077 51.72814,-179.46731 51.74634,-179.46502 51.74934,-179.46183 51.75378,-179.45943 51.75736,-179.45560 51.76273,-179.44845 51.77186,-179.43419 51.78977,-179.41259 51.81643,-179.41001 51.81941,-179.40751 51.82223,-179.40497 51.82505,-179.40242 51.82796,-179.39981 51.83095,-179.39734 51.83398,-179.39499 51.83709)'::geometry, ARRAY['2017-01-15 09:06:39'::timestamp,'2017-01-15 09:13:39','2017-01-15 09:14:48','2017-01-15 09:16:28','2017-01-15 09:17:48','2017-01-15 09:19:48','2017-01-15 09:23:19','2017-01-15 09:30:28','2017-01-15 09:34:40','2017-01-15 09:36:59','2017-01-15 09:38:09','2017-01-15 09:39:18','2017-01-15 09:40:40','2017-01-15 09:47:38','2017-01-15 09:48:49','2017-01-15 21:18:30'], '{"leafcount": 16, "attributes" : {"heading" : {"type": "float", "length": 4, "nullable" : false,"value": [23.0,23.0,23.0,23.0,21.0,21.0,72.0,72.0,72.0,72.0,73.0,74.0,73.0,73.0,73.0,73.0]}}}' ),'heading');
 st_attrfloataverage
---------------------
             53.8125
(1 row)
```

8.7.12. ST_leafType

获得轨迹的叶面类型。

语法

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

参数

参数名称	描述
traj	需要获得类型的轨迹。

目前只支持ST_POINT类型。

示例

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

8.7.13. ST_leafCount

获得轨迹的叶子数量（轨迹点个数）。

语法

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

参数

参数名称	描述
traj	需要获得类型的轨迹。

示例

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

8.7.14. ST_duration

获得该轨迹的持续时间。

语法

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

参数

参数名称	描述
traj	轨迹数据。

示例

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

8.7.15. ST_timeAtPoint

获取移动轨迹通过某位置的时间点集合。

语法

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

参数

参数名称	描述
traj	轨迹对象。
g	点位置。

示例

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

8.7.16. ST_pointAtTime

获取指定时间位置的几何对象。

语法

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

参数

参数名称	描述
traj	轨迹对象。
t	指定的时间点。

返回点对象。

示例

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

8.7.17. ST_velocityAtTime

获取指定时间位置的速度属性。

语法

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

参数

参数名称	描述
traj	轨迹对象。
t	指定的时间点。

示例

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

8.7.18. ST_accelerationAtTime

获取指定时间位置的加速度属性。

语法

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

参数

参数名称	描述
traj	轨迹对象。
t	指定的时间点。

示例

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

8.7.19. ST_timeToDistance

输出时间到欧氏距离的函数，以折线输出，横坐标为时间，纵坐标为距离。

语法

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

参数

参数名称	描述
traj	轨迹对象。

示例

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

8.7.20. ST_timeAtDistance

从起始点移动指定距离后所在的时间点。

语法

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

参数

参数名称	描述
traj	轨迹对象。
d	距离。

返回的是 timestamp的数组，有可能存在多个点到起始点的距离一致。

示例

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

8.7.21. ST_cumulativeDistanceAtTime

从起点出发到指定时间点累计的位移长度。

语法

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

参数

参数名称	描述
traj	轨迹对象。
t	时间点。

示例

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

8.7.22. ST_timeAtCumulativeDistance

从起点出发位移到指定长度时所处的时间点。

语法

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

参数

参数名称	描述
traj	轨迹对象。
d	累计长度。

示例

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

8.7.23. ST_subTrajectory

根据时间段截取子段。

语法

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

参数

参数名称	描述
traj	轨迹对象。
starttime	开始时间。

参数名称	描述
endtime	结束时间。
range	时间段。

示例

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

8.7.24. ST_subTrajectorySpatial

指定时间段的轨迹几何。

语法

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

参数

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

示例

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

8.7.25. ST_samplingInterval

获取采样间隔。

语法

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

参数

参数名称	描述
traj	轨迹对象。

示例

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

8.7.26. ST_trajAttrsAsText

获得作为文本类型的轨迹属性数组。

语法

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

参数

参数名称	描述
traj	轨迹对象。
attr_name	属性名称。

示例

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

8.7.27. ST_trajAttrsAsInteger

获得作为文本类型的轨迹属性数组。

语法

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

参数

参数名称	描述
traj	轨迹对象。
attr_name	属性名称。

示例

```
With traj AS (
 select '{"trajectory":{"version":1,"type":"STPOINT","leafcount":2,"start_time":"2010-01-01 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,null]},"speed":{"type":"float","length":8,"nullable":true,"value":[null,1.0]},"angel":{"type":"string","
length":64,"nullable":true,"value":["test",null]}, "tngel2":{"type":"timestamp","length":8,"nullable":true,"v
alue":["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)
```

8.7.28. ST_trajAttrsAsDouble

获得作为文本类型的轨迹属性数组。

语法

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

参数

参数名称	描述
traj	轨迹对象。
attr_name	属性名称。

示例

```
With traj AS (
 select '{"trajectory":{"version":1,"type":"STPOINT","leafcount":2,"start_time":"2010-01-01 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,null]},"speed":{"type":"float","length":8,"nullable":true,"value":[null,1.0]},"angel":{"type":"string","
length":64,"nullable":true,"value":["test",null]}, "tngel2":{"type":"timestamp","length":8,"nullable":true,"v
alue":["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)
```

8.7.29. ST_trajAttrsAsBool

获得作为文本类型的轨迹属性数组。

语法

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

参数

参数名称	描述
traj	轨迹对象。
attr_name	属性名称。

示例

```
With traj AS (
 select '{"trajectory":{"version":1,"type":"STPOINT","leafcount":2,"start_time":"2010-01-01 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,null]},"speed":{"type":"float","length":8,"nullable":true,"value":[null,1.0]},"angel":{"type":"string","
length":64,"nullable":true,"value":["test",null]}, "tngel2":{"type":"timestamp","length":8,"nullable":true,"v
alue":["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)
```

8.7.30. ST_trajAttrsAsTimestamp

获得作为时间戳类型的轨迹属性数组。

语法

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

参数

参数名称	描述
traj	轨迹对象。
attr_name	属性名称。

示例

```
With traj AS (
 select '{"trajectory":{"version":1,"type":"STPOINT","leafcount":2,"start_time":"2010-01-01 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,null]},"speed":{"type":"float","length":8,"nullable":true,"value":[null,1.0]},"angel":{"type":"string","
length":64,"nullable":true,"value":["test",null]}, "tngel2":{"type":"timestamp","length":8,"nullable":true,"v
alue":["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}
```

8.7.31. ST_attrIntFilter

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

语法

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

参数

参数名称	描述
traj	轨迹对象attr。
attr_field_name	指定的属性名称。
operator	过滤符，`'='`，`'!='`，`'>'`，`'<'`，`'>='`，`'<='`，`'[]'`，`'()'`，`'[)'`，`'(]'`。
value、value1	属性固定值下限。

参数名称	描述
value2	属性固定值上限。

描述

只支持类型为integer的属性字段。

示例

```
create table traj(id integer, traj trajectory);
insert into traj(traj) values(ST_makeTrajectory('STPOINT'::leaftype, 'LINESTRING(-179.48077 51.72814,-179.46731 51.74634,-179.46502 51.74934,-179.46183 51.75378,-179.45943 51.75736,-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}
```

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

8.7.32. ST_attrFloatFilter

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

语法

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

参数

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

描述

只支持类型为float的属性字段。

示例

```
create table traj(id integer, traj trajectory);
insert into traj values(2,ST_makeTrajectory('STPOINT'::leaftype, 'LINESTRING(-179.48077 51.72814,-179.46731 51.74634,-179.46502 51.74934,-179.46183 51.75378,-179.45943 51.75736,-179.45560 51.76273,-179.44845 51.77186,-179.43419 51.78977,-179.41259 51.81643,-179.41001 51.81941,-179.40751 51.82223,-179.40497 51.82505,-179.40242 51.82796,-179.39981 51.83095,-179.39734 51.83398,-179.39499 51.83709)'::geometry, ARRAY['2017-01-15 09:06:39'::timestamp,'2017-01-15 09:14:48','2017-01-15 09:13:39','2017-01-15 09:16:28','2017-01-15 09:19:48','2017-01-15 09:17:48','2017-01-15 09:23:19','2017-01-15 09:34:40','2017-01-15 09:30:28','2017-01-15 09:36:59','2017-01-15 09:38:09','2017-01-15 09:39:18','2017-01-15 09:40:40','2017-01-15 09:47:38','2017-01-15 21:18:30','2017-01-15 09:48:49'], '{"leafcount": 16, "attributes" : {"heading" : {"type": "float", "length": 8, "nullable" : false,"value":[23.0,23.0,23.0,23.0,21.0,21.0,72.0,72.0,72.0,72.0,73.0,74.0,73.0,73.0,73.0,73.0]}}}'));
select st_attrFloatFilter(traj, 'heading', '>', 23.0) from traj where id = 2;
    st_attrfloatfilter
---------------------------------
{72,72,72,72,73,74,73,73,73,73}
(1 row)
```

8.7.33. ST_attrTimestampFilter

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

语法

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

参数

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

示例

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

8.7.34. ST_attrNullFilter

指定轨迹属性字段，过滤出值为空的轨迹点，返回由这些轨迹点构成的新轨迹。

语法

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

参数

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

描述

支持所有类型的属性字段。

示例

```
select st_attrNullFilter(ST_makeTrajectory('STPOINT'::leaftype, 'LINESTRING(-179.48077 51.72814,-179.46731 51.74634,-179.46502 51.74934,-179.46183 51.75378,-179.45943 51.75736,-179.45560 51.76273,-179.44845 51.77186,-179.43419 51.78977,-179.41259 51.81643,-179.41001 51.81941,-179.40751 51.82223,-179.40497 51.82505,-179.40242 51.82796,-179.39981 51.83095,-179.39734 51.83398,-179.39499 51.83709)'::geometry, ARRAY['2017-01-15 09:06:39'::timestamp,'2017-01-15 09:13:39','2017-01-15 09:14:48','2017-01-15 09:16:28','2017-01-15 09:17:48','2017-01-15 09:19:48','2017-01-15 09:23:19','2017-01-15 09:30:28','2017-01-15 09:34:40','2017-01-15 09:36:59','2017-01-15 09:38:09','2017-01-15 09:39:18','2017-01-15 09:40:40','2017-01-15 09:47:38','2017-01-15 09:48:49','2017-01-15 21:18:30'], '{"leafcount": 16, "attributes" : {"heading" : {"type": "float", "length": 4, "nullable" : true,"value": [23.0,23.0,23.0,null,21.0,21.0,null,72.0,72.0,null,73.0,74.0,73.0,73.0,null,73.0]}}}' ),'heading');
                st_attrnullfilter
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
 {"trajectory":{"version":1,"type":"STPOINT","leafcount":4,"start_time":"2017-01-15 09:16:28","end_time":"2017-01-15 09:48:49","spatial":"LINESTRING(-179.46183 51.75378,-179.44845 51.77
186,-179.41001 51.81941,-179.39734 51.83398)","timeline":["2017-01-15 09:16:28","2017-01-15 09:23:19","2017-01-15 09:36:59","2017-01-15 09:48:49"],"attributes":{"leafcount":4,"heading":
{"type":"float","length":4,"nullable":true,"value":[null,null,null,null]}}}
(1 row)
```

8.7.35. ST_attrNotNullFilter

指定轨迹属性字段，过滤出值不为空的轨迹点，返回由这些轨迹点构成的新轨迹。

语法

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

参数

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

描述

支持所有类型的属性字段。

示例

```
select st_attrNotNullFilter(ST_makeTrajectory('STPOINT'::leaftype, 'LINESTRING(-179.48077 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)
```

8.7.36. ST_trajAttrsMeanMax

根据MEAN-MAX算法，计算出每个时间段内的均值的最大值。

语法

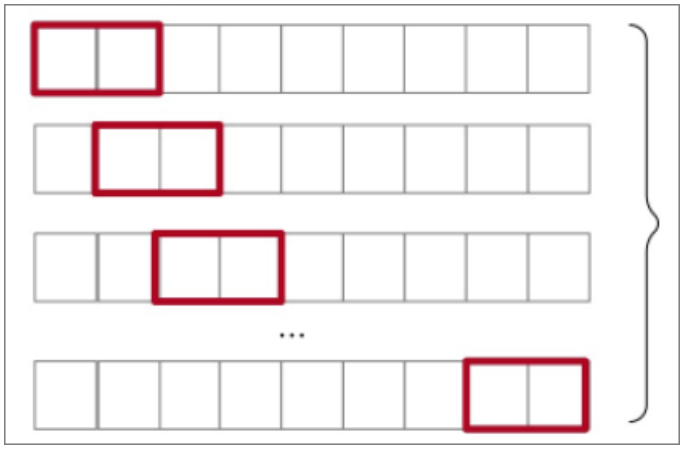
```
SETOF recrod ST_trajAttrsMeanMax(trajectory traj, cstring attr_field_name, out interval duration, out float8 max);
```

参数

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

描述

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



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

示例

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

8.8. 外包框类型和处理函数

8.8.1. ST_MakeBox

获取指定类型在指定时间段内的外包框。

语法

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

参数

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

描述

外包框即包含某个时空物体的最小多维矩形。

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

示例

```
WITH geom AS(
  SELECT ('POLYGON((12.7243236691148 4.35238368367118,12.9102992732078 1.49748113937676,12.592659
2946053 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) f
rom geom;
                st_makebox                |                                st_makebox
------------------------------------------------------------+-----------------------------------------------------------
-----------------------------------------
 BOX2D(12.0197572708 1.49748110771,12.9102993011 4.35238409042) | BOX2DT(12.0197572708 1.497481107
71 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 2000-0
1-01 00:00:30.000001)
```

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

直接写入值来构建Box。

语法

```
boxndf ST_MakeBoxZ(float8 zmin, float8 zmax);
boxndf ST_MakeBoxT(timestamp tmin, timestamp tmax);
boxndf ST_MakeBox2D(float8 xmin, float8 ymin, float8 xmax, float8 ymax);
boxndf ST_MakeBox2DT(float8 xmin, float8 ymin, timestamp tmin, float8 xmax,float8 yax, 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, floa
t8 zmax, timestamp tmax);
```

参数

参数名称	描述
xmin	外包框的x轴下界。
xmax	外包框的x轴上界。

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

描述

根据函数名和指定的参数构建外包框。

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

示例

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

8.8.3. ST_BoxndfToGeom

将外包框类型转化为Geometry类型。

语法

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

参数

参数名称	描述
box	指定的外包框。

描述

将外包框类型转化为Geometry类型。

- 如果外包框有z维度，将转化为三维Geometry类型。

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

示例

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

8.8.4. ST_Has{xy|z|t}

确定指定的外包框是否包含某些维度。

语法

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

参数

参数名称	描述
box	指定的外包框。

描述

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

示例

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

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

获取BoxNDF类型中指定维度的最小值。

语法

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

参数

参数名称	描述
box	指定的外包框。

描述

获取外包框类型中指定维度的最小值。如果外包框不包含指定维度时：

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

示例

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

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

获取BoxNDF类型中指定维度的最大值。

语法

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

参数

参数名称	描述
box	指定的外包框。

描述

获取外包框类型中指定维度的最大值。如果外包框不包含指定维度时：

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

示例

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

8.8.7. ST_ExpandSpatial

将外包框的空间范围扩大至指定数值。

语法

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

参数

参数名称	描述
box	需要扩大的外包框。
len	需要扩大的长度。

描述

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

示例

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

8.9. 外包框处理算子

8.9.1. 基本概念

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

算子类别标记

算子类别标记目前分为三种：

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

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

维度标记

维度标记目前分为两种：

- z维度标记：/
- t维度标记：#

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

8.9.2. 相交类算子

判断左侧对象和右侧对象的外包框是否在指定维度上相交。

语法

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

参数

参数名称	描述
算子左侧参数	相交对象。

参数名称	描述
算子右侧参数	相交对象。

描述

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

支持的相交类算子如下：

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

示例

```
WITH box AS(
  SELECT ST_MakeBox3d(0,0,0,5,5,5) a,
      ST_MakeBox3dt(6,6, 3,'2010-04-12 00:00:00',8,8,5,'2013-02-01 00:00:00') b
)
SELECT a /&/ b AS OpZ, a #&# b AS OpT, a && b AS Op2D, a &/& b AS Op3d, a &#& b AS Op2DT, a &/#& b AS Op3
DT from box;
 opz | opt | op2d | op3d | op2dt | op3dt
-----+-----+------+------+-------+-------
t  | t  | f  | f  | f  | f
```

8.9.3. 包含算子

判断左侧对象的外包框是否在指定维度上包含右侧对象的外包框。

语法

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

参数

参数名称	描述
算子左侧参数	包含对象。
算子右侧参数	被包含对象。

描述

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

支持的包含算子如下：

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

示例

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

8.9.4. 被包含算子

判断右侧对象的外包框是否在指定维度上包含左侧对象的外包框。

语法

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

参数

参数名称	描述
算子左侧参数	包含对象。
算子右侧参数	被包含对象。

描述

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

支持的被包含算子如下：

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

示例

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

8.10. 空间关系判断

8.10.1. ST_intersects

指定时间区间的轨迹段和几何图形空间是否相交。

语法

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

参数

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

示例

```
Select ST_intersects(traj, '2010-1-1 13:00:00', '2010-1-1 14:00:00', 'LINESTRING(0 0, 5 5, 9 9)'::geometry) from t
raj_table;
```

8.10.2. ST_equals

指定时间区间的轨迹段和几何图形空间是否相同。

语法

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

参数

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

示例

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

8.10.3. ST_distanceWithin

指定时间区间的轨迹段离给定几何对象在指定距离之内。

语法

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

参数

参数名称	描述
traj	轨迹对象。
t1	开始时间。

参数名称	描述
t2	结束时间。
range	时间段。
g	几何对象。
d	距离。

示例

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

8.11. 空间处理

8.11.1. ST_intersection

给定时间段的轨迹和给定的几何对象执行相交处理，返回相交处理后的轨迹对象。

语法

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

参数

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

描述

如果轨迹几何对象和几何对象有多个交点，则返回多个子轨迹。

示例

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

8.11.2. ST_difference

给定时间段的轨迹和给定的几何对象执行相差处理，返回相交处理后的轨迹对象。

语法

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

参数

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

示例

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

8.12. 空间统计

8.12.1. ST_nearestApproachPoint

指定时间段的轨迹中找到与给定几何对象最邻近点信息。

语法

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

参数

参数名称	描述
traj	轨迹对象。
t1	开始时间。
t2	结束时间。

参数名称	描述
range	时间段。
g	几何对象。

示例

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

8.12.2. ST_nearestApproachDistance

指定时间段的轨迹中找到与给定几何对象最近距离。

语法

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

参数

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

示例

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

8.13. 时空关系判断

8.13.1. ST_intersects

指定时间区间的轨迹段1和轨迹段2是否相交。

语法

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

参数

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

示例

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

8.13.2. ST_equals

指定时间区间的轨迹段1和轨迹段2空间是否相等。

语法

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

参数

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

示例

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

8.13.3. ST_distanceWithin

指定时间区间的轨迹段1离给定轨迹段2是否在指定距离之内。

语法

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

参数

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

示例

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

8.13.4. ST_durationWithin

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

语法

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

参数

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

参数名称	描述
i	时间间隔。

描述

如果轨迹多次经过相同的点，任意一个时间符合要求就认为符合要求。

示例

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

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

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

语法

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

参数

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

描述

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

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

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

示例

```
WITH traj AS(
  SELECT ('{"trajectory":{"version":1,"type":"STPOINT","leafcount":6,"start_time":"2000-01-01 03:15:42","
end_time":"2000-01-01 05:16:43",' ||
      '"spatial":"LINESTRING(2 2 0,33.042158099636 36.832684322819 0,47.244002354518 47.230026333034 0,
64.978971942887 60.618813472986 0,77.621717839502 78.012496630661 0,80 78 0)",' ||
      '"timeline":["2000-01-01 03:15:42","2000-01-01 03:39:54","2000-01-01 04:04:06","2000-01-01 04:28:18","
2000-01-01 04:52:31","2000-01-01 05:16:43"]}}')::trajectory a,
     ('{"trajectory":{"version":1,"type":"STPOINT","leafcount":4,"start_time":"2000-01-01 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
```

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

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

语法

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

参数

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

描述

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

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

示例

```
EXPLAIN SELECT count(*) FROM sqltr_test_trajs WHERE ST_2DIntersects_IndexLeft(traj, ST_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)
```

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

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

语法

```
bool ST_{2D|3D}DWithin(geometry geom, trajectory traj, float8 dist);
bool ST_{2D|3D}DWithin(trajectory traj, geometry geom, float8 dist);
bool ST_{2D|3D}DWithin(geometry geom, trajectory traj, timestamp ts, timestamp te, float8 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);
```

参数

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

描述

- 对于2D和3D的DWithin操作，我们判断两者在二维或三维空间上的投影（即对应的geometry类型）是否距离小于等于给定值。
- 对于2DT和3DT的DWithin操作，我们判断两者是否在某一时间点上在指定维度上的距离小于等于过给定值。

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

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

示例

```
WITH traj AS(
  Select ST_makeTrajectory('STPOINT', 'LINESTRING(0 0 10, 50 0 10, 100 0 10)'::geometry,
                ('[' || ST_PGEpochToTS(0) || ',' || ST_PGEpochToTS(100) || ')')::tsrange,
                '{"leafcount":3,"attributes":{"velocity": {"type": "integer", "length": 2,"nullable" : true,"value":
[120,130,140]}, "accuracy": {"type": "float", "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
```

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

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

语法

```
bool ST_{2D|2DT|3D|3DT}DWithin_IndexLeft(trajectory traj1, trajectory traj2, float8 dist);
bool ST_{2D|2DT|3D|3DT}DWithin_IndexLeft(trajectory traj1, trajectory traj2, timestamp ts, timestamp te, fl
oat8 dist);
```

参数

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

描述

当 ST_ndDWithin(...) 操作涉及两个轨迹列时（即调用 ST_{2D|2DT|3D|3DT}DWithin(traiectorv trai1. traiectorv trai2. float8 dist) 或 ST_{2D|2DT|3D|3DT}DWithin(trajectory traj1, trajectory traj2, timestamp ts, timestamp te, float8 dist) 函数时），数据库无法判断应当使用哪个列的索引，因此会带来额外的计算开销。

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

示例

```
EXPLAIN SELECT count(*) FROM sqltr_test_trajs WHERE ST_3DTDWithin_IndexLeft(traj, ST_makeTrajectory('STPOINT', 'LINESTRING(-70 -30, -72 -34, -73 -35)'::geometry, '[2000-01-01 00:01:30, 2000-01-01 00:15:00)'::tsrange,
 '{"leafcount":3,"attributes":{"velocity": {"type": "integer", "length": 2,"nullable" : true,"value": [120,130,140]}, "accuracy": {"type": "float", "length": 4, "nullable" : false,"value": [120,130,140]}, "bearing": {"type": "float", "length": 8, "nullable" : false,"value": [120,130,140]}, "acceleration": {"type": "string", "length": 20, "nullable" : true,"value": ["120","130","140"]}, "active": {"type": "timestamp", "nullable" : false,"value": ["Fri Jan 01 11:35:00 2010", "Fri Jan 01 12:35:00 2010", "Fri Jan 01 13:30:00 2010"]}}, "events": [{"2" : "Fri Jan 02 15:00:00 2010"}, {"3" : "Fri Jan 02 15:30:00 2010"}]}'), 2);
Aggregate  (cost=4341.89..4341.90 rows=1 width=8)
  -> Bitmap Heap Scan on sqltr_test_trajs  (cost=103.31..4340.56 rows=529 width=0)
      Recheck Cond: ('BOX2DT(-75 -37 2000-01-01 00:01:29.999994,-68 -28 2000-01-01 00:15:00)'::boxndf &/#& traj)
      Filter: _st_3dtdwithin(traj, '{"trajectory":{"version":1,"type":"STPOINT","leafcount":3,"start_time":"2000-01-01 00:01:30","end_time":"2000-01-01 00:15:00","spatial":"LINESTRING(-70 -30,-72 -34,-73 -35)","timeline":["2000-01-01 00:01:30","2000-01-01 00:08:15","2000-01-01 00:15:00"],"attributes":{"leafcount":3,"velocity":{"type":"integer","length":2,"nullable":true,"value":[120,130,140]},"accuracy":{"type":"float","length":4,"nullable":false,"value":[120.0,130.0,140.0]},"bearing":{"type":"float","length":8,"nullable":false,"value":[120.0,130.0,140.0]},"acceleration":{"type":"string","length":20,"nullable":true,"value":["120","130","140"]},"active":{"type":"timestamp","length":8,"nullable":false,"value":["2010-01-01 11:35:00","2010-01-01 12:35:00","2010-01-01 13:30:00"]}},"events":[{"2":"2010-01-02 15:00:00"},{"3":"2010-01-02 15:30:00"}]}}'::trajectory, '2'::double precision)
      -> Bitmap Index Scan on sqltr_test_traj_gist_2dt  (cost=0.00..103.18 rows=1586 width=0)
          Index Cond: (traj &/#& 'BOX2DT(-75 -37 2000-01-01 00:01:29.999994,-68 -28 2000-01-01 00:15:00)'::boxndf)
```

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

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

语法

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

参数

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

描述

判断第一个参数是否包含第二个参数。

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

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

示例

```
WITH traj AS(
  Select ST_makeTrajectory('STPOINT', 'LINESTRING(0 0, 50 50, 100 100)'::geometry,
                ('[' || ST_PGEpochToTS(0) || ',' || ST_PGEpochToTS(100) || ')'::tsrange,
                '{"leafcount":3,"attributes":{"velocity": {"type": "integer", "length": 2,"nullable" : true,"value":
[120,130,140]}, "accuracy": {"type": "float", "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
```

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

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

语法

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

参数

参数名称	描述
geom	需要判断的几何对象。

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

描述

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

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

示例

```
WITH traj AS(
  SELECT ('{"trajectory":{"version":1,"type":"STPOINT","leafcount":6,"start_time":"2000-01-01 03:15:42","
end_time":"2000-01-01 05:16:43",' ||
       '"spatial":"LINESTRING(2 2 0,33.042158099636 36.832684322819 0,47.244002354518 47.230026333034 0,
64.978971942887 60.618813472986 0,77.621717839502 78.012496630661 0,80 78 0)",' ||
       '"timeline":["2000-01-01 03:15:42","2000-01-01 03:39:54","2000-01-01 04:04:06","2000-01-01 04:28:18","
2000-01-01 04:52:31","2000-01-01 05:16:43"]}}')::trajectory a,
      'LINESTRING(2 2 0,33.042158099636 36.832684322819 0,47.244002354518 47.230026333034 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
```

8.14. 时空处理

8.14.1. ST_intersection

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

语法

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

参数

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

示例

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

8.15. 时空统计

8.15.1. ST_nearestApproachPoint

指定时间段的轨迹1和轨迹2中找到最邻近点信息。

语法

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

参数

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

示例

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

8.15.2. ST_nearestApproachDistance

指定时间段的轨迹1和轨迹2中找到最邻近距离。

语法

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

参数

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

示例

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

8.16. 距离测量

8.16.1. ST_length

获取一条轨迹的行程总长度，单位为米。

语法

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

参数

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

描述

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

示例

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

8.16.2. ST_euclideanDistance

计算两个轨迹对象之间的欧几里得距离。

语法

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

参数

参数名称	描述
traj1	轨迹对象1。
traj2	轨迹对象2。

描述

距离已经进行了标准化处理。

示例

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

8.16.3. ST_mdistance

计算轨迹对象中所有相同的时间点的欧几里得距离。

语法

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

参数

参数名称	描述
traj1	轨迹对象1。

参数名称	描述
traj2	轨迹对象2。

描述

未经过标准化处理。

示例

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

8.17. 相似度分析

8.17.1. ST_lcsSimilarity

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

语法

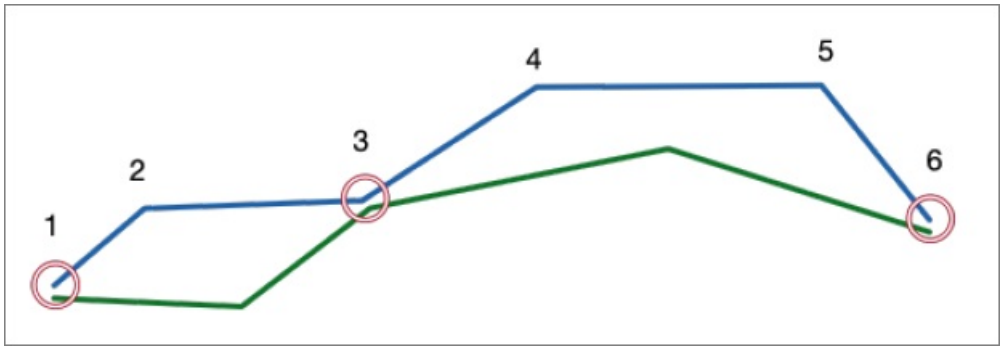
```
integer ST_lcsSimilarity(trajectory traj1, trajectory traj2, float8 dist, distanceUnit unit default 'M' );
integer ST_lcsSimilarity(trajectory traj1, trajectory traj2, float8 dist, interval lag, distanceUnit unit default '
M');
```

参数

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

描述

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



上图中轨迹点1，3， 6 符合要求，返回为 3。

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

示例

```
With traj AS (
  Select ST_makeTrajectory('STPOINT', 'LINESTRINGZ(114.000528 33.588163 54.87 , 114.000535 33.588235 54
.85 , 114.000447 33.588272 54.69 , 114.000348 33.588287 54.73 , 114.000245 33.588305 55.26 , 114.000153 33.58
8305 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 33.5883
05 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 1
1:33:09','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 33.58
8305 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 33.5883
05 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 1
1:34:15','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)
```

8.17.2. ST_lcsDistance

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

语法

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

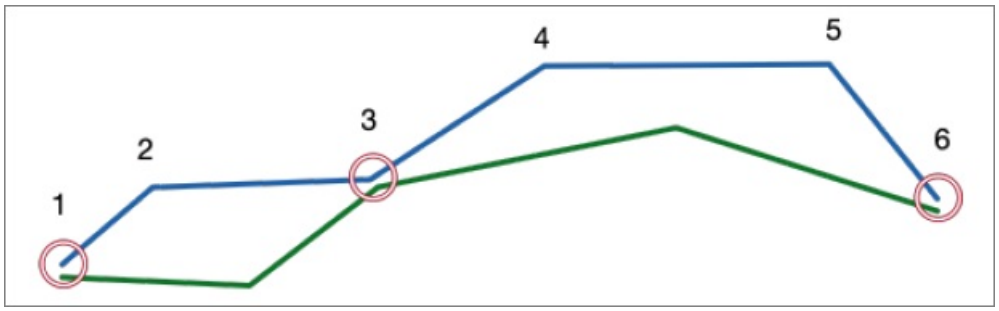
参数

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

描述

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

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



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

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

示例

```
With traj AS (
  Select ST_makeTrajectory('STPOINT', 'LINESTRINGZ(114.000528 33.588163 54.87 , 114.000535 33.588235 54
.85 , 114.000447 33.588272 54.69 , 114.000348 33.588287 54.73 , 114.000245 33.588305 55.26 , 114.000153 33.58
8305 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 33.5883
05 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 1
1:33:09','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 33.58
8305 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 33.5883
05 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 1
1:34:15','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 33.58
8305 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 33.5883
05 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 1
1:34:15','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)
```

8.17.3. ST_lcsSubDistance

计算LCSS轨迹段与traj1在这段轨迹上的距离。

语法

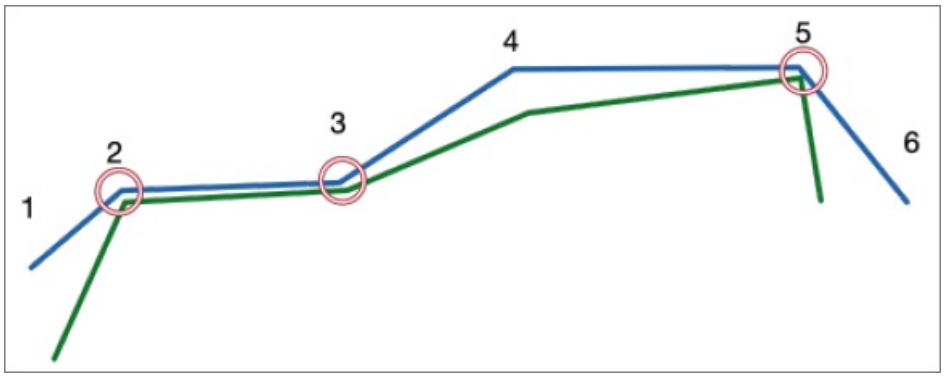
```
float8 ST_lcsSubDisatance(trajectory traj1, trajectory traj2, float8 dist, distanceUnit unit default 'M');
float8 ST_lcsSubDisatance(trajectory traj1, trajectory traj2, float8 dist, interval lag, distanceUnit unit defaul
t 'M');
```

参数

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

描述

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



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

数值越小表明LCSS轨迹与轨迹1的相似度越高。

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

示例

```
With traj AS (
  Select ST_makeTrajectory('STPOINT', 'LINESTRINGZ(114.000528 33.588163 54.87 , 114.000535 33.588235 54
.85 , 114.000447 33.588272 54.69 , 114.000348 33.588287 54.73 , 114.000245 33.588305 55.26 , 114.000153 33.58
8305 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 33.5883
05 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 1
1:33:09','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 33.58
8305 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 33.5883
05 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 1
1:33:09','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)
```

8.17.4. ST_JaccardSimilarity

计算轨迹或子轨迹的Jaccard相似系数（Jaccard index）。

语法

```
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') ;
```

参数

参数名称	描述
tr1	轨迹对象1。
tr2	轨迹对象2。

参数名称	描述
tol_dist	空间容差，单位为米。
unit	描述距离计算方式的JSON字符串。
tol_time	时间容差。默认为NULL，当值为NULL或负值时表示只进行空间匹配不考虑时间。
ts	开始时间。默认为 `-infinity` ，只关注在起止时间之间的子轨迹。
te	结束时间。默认为 `infinity` ，只关注在起止时间之间的子轨迹。

unit参数说明如下。

参数名称	类型	默认值	说明
Projection	string	无	重新投影的目标坐标系。取值： • auto：根据经纬度坐标动态选择最合适的坐标系（Lambert Azimuthal、UTM等），并投影到此坐标系进行计算。不需要额外指定Unit，单位为米。 • srid：需要重投影到的空间参考系ID。 **说明** 不传本参数时，依照原有坐标系计算。
Unit	string	null	度量单位。取值： • null：即无单位，直接按照轨迹点坐标的欧式距离进行计算。 • M：按照所在空间参考系基于的单位进行距离计算，通常为米。
useSpheroid	bool	true	是否使用椭球体计算。当Unit设置为M时，可以指定此项。取值： • true：使用椭球体并计算更精确的距离。 • false：使用球体表示地球并计算近似的距离。

返回参数说明如下。

参数名称	类型	说明
nleaf1	int	轨迹1与轨迹2相交的点数量。
nleaf2	int	轨迹2与轨迹1相交的点数量。 **说明** 与nleaf1不一定相同，例如可能轨迹1两次经过轨迹2的同一个点，则nleaf1为1，nleaf2为2。

参数名称	类型	说明
inter1	int	轨迹1到轨迹2的距离同时满足时间容差和空间容差的点数量。
inter2	int	轨迹2到轨迹1的距离同时满足时间容差和空间容差的点数量。
jaccard_lower	double	Jaccard距离下限。计算公式：`inter/(nleaf1+nleaf2-inter)`。 **说明** inter为inter1和inter2中较小的值。
jaccard_upper	double	Jaccard距离上限。计算公式：`inter/(nleaf1+nleaf2-inter)`。 **说明** inter为inter1和inter2中较大的值。

描述

两个集合的Jaccard index指两个集合交集的元素数量除以两个集合并集的元素数量。对于两条轨迹，我们扩展了Jaccard index的定义，分别计算在轨迹1与轨迹2相交的点数量，以及在轨迹2上与轨迹1相交的点数量，并通过上文jaccard_lower和jaccard_upper的公式计算距离。

示例

```
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)
```

8.18. 索引方式

8.18.1. GisT索引

对轨迹数据列创建GisT索引。

语法

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

- index_name：索引名，可以省略。
- table_name：表名。
- traj_col：轨迹列名。
- operator_family：指定索引所使用的算子族，可以省略，默认值为trajgist_ops_multi，可通过ganos.trajectory.index_split_config参数调整。

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

支持算子族

当前支持7个索引的算子族，说明如下。

算子族名称	描述
trajgist_ops_z	对z轴范围建立索引，支持仅包含z轴的查询。
trajgist_ops_t	对t轴范围建立索引，支持仅包含t轴的查询。
trajgist_ops_2d	对x、y轴范围建立索引，支持仅包含x、y轴信息的查询。
trajgist_ops_2dt	对x、y、t轴范围建立索引，支持二维、时间，以及混合查询。
trajgist_ops_3d	对x、y、z轴范围建立索引，支持二维、三维、z轴查询。
trajgist_ops_3dt	对x、y、z、t轴范围建立索引，支持以上全部查询。
trajgist_ops_multi	建立多外包框架索引，以加速查询，但建立更慢并占用更多空间。

示例

- 建立时间维索引。

```
CREATE INDEX on table_name USING GIST(traj_col trajgist_ops_t);
```

- 建立二维索引。

```
CREATE INDEX on table_name USING GIST(traj_col trajgist_ops_2d);
```

- 建立二维时空复合索引。

```
CREATE INDEX on table_name USING GIST(traj_col trajgist_ops_2dt);
```

8.18.2. TrajGisT索引

TrajGisT索引是GisT索引的扩展。

背景信息

在GisT基础上，TrajGisT提供额外两点优化：

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

语法

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

- index_name：索引名，可以省略。
- table_name：表名。
- traj_col：轨迹列名。

- operator_family：指定索引所使用的算子族，可以省略，默认值为trajgist_ops_multi，可通过ganos.trajectory.index_split_config参数调整。

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

支持算子族

当前支持7个索引的算子族，说明如下。

算子族名称	描述
trajgist_ops_z	对z轴范围建立索引，支持所有包含z轴的查询。
trajgist_ops_t	对t轴范围建立索引，支持所有包含t轴的查询。
trajgist_ops_2d	对x、y轴范围建立索引，支持所有包含x、y轴信息的查询。
trajgist_ops_2dt	对x、y、t轴范围建立索引，支持所有包含x、y、t轴的查询。
trajgist_ops_3d	对x、y、z轴范围建立索引，支持所有包含x、y、z轴的查询。
trajgist_ops_3dt	对x、y、z、t轴范围建立索引，支持以上全部查询。
trajgist_ops_multi	建立多外包框索引，以加速查询，但建立更慢并占用更多空间。

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

8.19. 变量

8.19.1. ganos.trajectory.attr_string_length

设置字符串类型属性默认长度。

描述

`attr_string_length` 为integer类型。

示例

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

8.19.2. ganos.trajectory.index_split_config

设置使用GisT索引时，切割轨迹所用的规则。

描述

`index_split_config` 为string类型。字符串格式与ST_Split格式相同。

默认值

```
"{\"cut_point.even_divide\":10}"
```

9.GeomGrid SQL参考

9.1. 使用简介

GeomGrid是Ganos GeomGrid扩展中用于表示一个网格对象的数据类型。

数据类型

- GeomGrid支持以下数据类型到GeomGrid的转换：
  - text
  - bytea
- GeomGrid支持从GeomGrid到以下数据类型的转换：
  - text
  - bytea
  - geometry
  - box

9.2. 输出

9.2.1. ST_AsText

将一个网格对象转换为指定规范的文本编码。

语法

```
text ST_AsText(geomgrid   grid,
        integer precision,
        text standard default 'GGER')
text[] ST_AsText(geomgrid[]   grid,
        integer precision,
        text standard default 'GGER')
```

参数

参数名称	描述
grid	需要输出的网格对象。
precision	精度级别，取值：0~31，-1表示使用默认精度级别。
standard	规范标准：GGER。自然资源部地球空间网格编码规则（GeoSpatial Grid Encoding Rule）。默认使用GGER。

描述

将一个网格对象按照指定的层级、精度以及编码规范输出。当指定的精度高于网格存储的精度时，不进行补零输出。

示例

```
--使用默认层级
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
--指定输出的层级
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
```

9.2.2. ST_AsBinary

将一个网格对象转换为二进制结构。

语法

```
bytea ST_AsBinary(geomgrid  grid)
```

参数

参数名称	描述
grid	需要输出的网格对象。

示例

```
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
```

9.2.3. ST_AsGeometry

返回以几何对象表示的网格对象范围。

语法

```
geometry ST_AsGeometry(geomgrid grid);
geometry[] ST_AsGeometry(geomgrid[] grid);
```

参数

参数名称	描述
grid	需要输出的网格对象。

示例

```
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))
```

9.2.4. ST_AsBox

从网格对象返回以box表示的网格范围。

语法

```
box ST_AsBox(geomgrid grid);
box[] ST_AsBox(geomgrid[] grid);
```

参数

参数名称	描述
grid	需要输出的网格对象。

示例

```
select st_asbox(
  st_gridfromtext('G0013103220310313'));
                    st_asbox
--------------------------------------------------------------------------------
(116.46666666666667,39.31666666666667),(116.4588888888889,39.30888888888889)
```

9.3. 输入

9.3.1. ST_GridFromText

将一个网格对象字符串转换为geomgrid数据类型。

语法

```
geomgrid ST_GridFromText(text  Code)
```

参数

参数名称	描述
Code	以字符串方式表达的网格码。

示例

```
select st_gridfromtext('G0013103220310313');
  st_gridfromtext
----------------------
 01011C1074271B120101
```

9.3.2. ST_GridFromBinary

从一个网格对象二进制转换为geomgrid数据类型。

语法

```
geomgrid ST_GridFromBinary(bytea  binary)
```

参数

参数名称	描述
binary	二进制表示的网格对象。

示例

```
select st_astext(
 st_gridfrombinary('\x01010c0f74271236'));
  st_astext
-------------------
 G001310322230230
```

9.4. 空间关系判断

9.4.1. ST_Intersects

查询网格对象代表的空间范围和几何对象是否相交。

语法

```
boolean ST_Intersects(geomgrid grid, geometry geom);
boolean ST_Intersects(geometry geom, geomgrid grid);
```

参数

参数名称	描述
grid	网格对象。
geom	几何对象。

描述

几何对象空间参考必须是CGC2000（SRID=4490）。

示例

```
select st_intersects(ST_gridfromtext('G0013310322133000013'),
    ST_geomfromtext('LINESTRING(122.48077 51.72814, 122.47416 51.73714)',4490));
 st_intersects
---------------
 t
```

9.4.2. ST_Contains

查询网格对象代表的空间范围和几何对象是否是包含关系。

语法

```
boolean ST_Contains(geomgrid grid, geometry geom);
boolean ST_Contains(geometry geom, geomgrid grid);
boolean ST_Contains(geomgrid grid1, geomgrid grid2);
```

参数

参数名称	描述
grid	网格对象。
geom	几何对象。

示例

```
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
```

9.4.3. ST_Within

查询网格对象代表的空间范围和几何对象是否是被包含关系。

语法

```
boolean ST_Within(geomgrid grid, geometry geom);
boolean ST_Within(geometry geom, geomgrid grid);
boolean ST_Within(geomgrid grid1, geomgrid grid2);
```

参数

参数名称	描述
grid	网格对象。
geom	几何对象。

描述

几何对象空间参考必须是CGC2000（SRID=4490）。

示例

```
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
```

9.5. 操作符

9.5.1. @>

查询网格对象代表的空间范围是否包含几何对象。

语法

```
boolean @>(geomgrid grid, geometry geom);
boolean @>(geometry geom, geomgrid grid);
boolean @>(geomgrid grid1, geomgrid grid2);
```

参数

参数名称	描述
grid	网格对象。
geom	几何对象。

描述

几何对象空间参考必须是CGC2000（SRID=4490）。

示例

```
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
```

9.5.2. <@

查询网格对象代表的空间范围是否被几何对象包含。

语法

```
boolean <@(geomgrid grid, geometry geom);
boolean <@(geometry geom, geomgrid grid);
boolean <@(geomgrid grid1, geomgrid grid2);
```

参数

参数名称	描述
grid	网格对象。
geom	几何对象。

描述

几何对象空间参考必须是CGC2000（SRID=4490）。

示例

```
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
```

9.6. 网格对象计算

9.6.1. ST_AsGrid

计算几何对象相交的所有网格对象。

语法

```
geomgrid[] ST_AsGrid(geometry geom, integer precision);
```

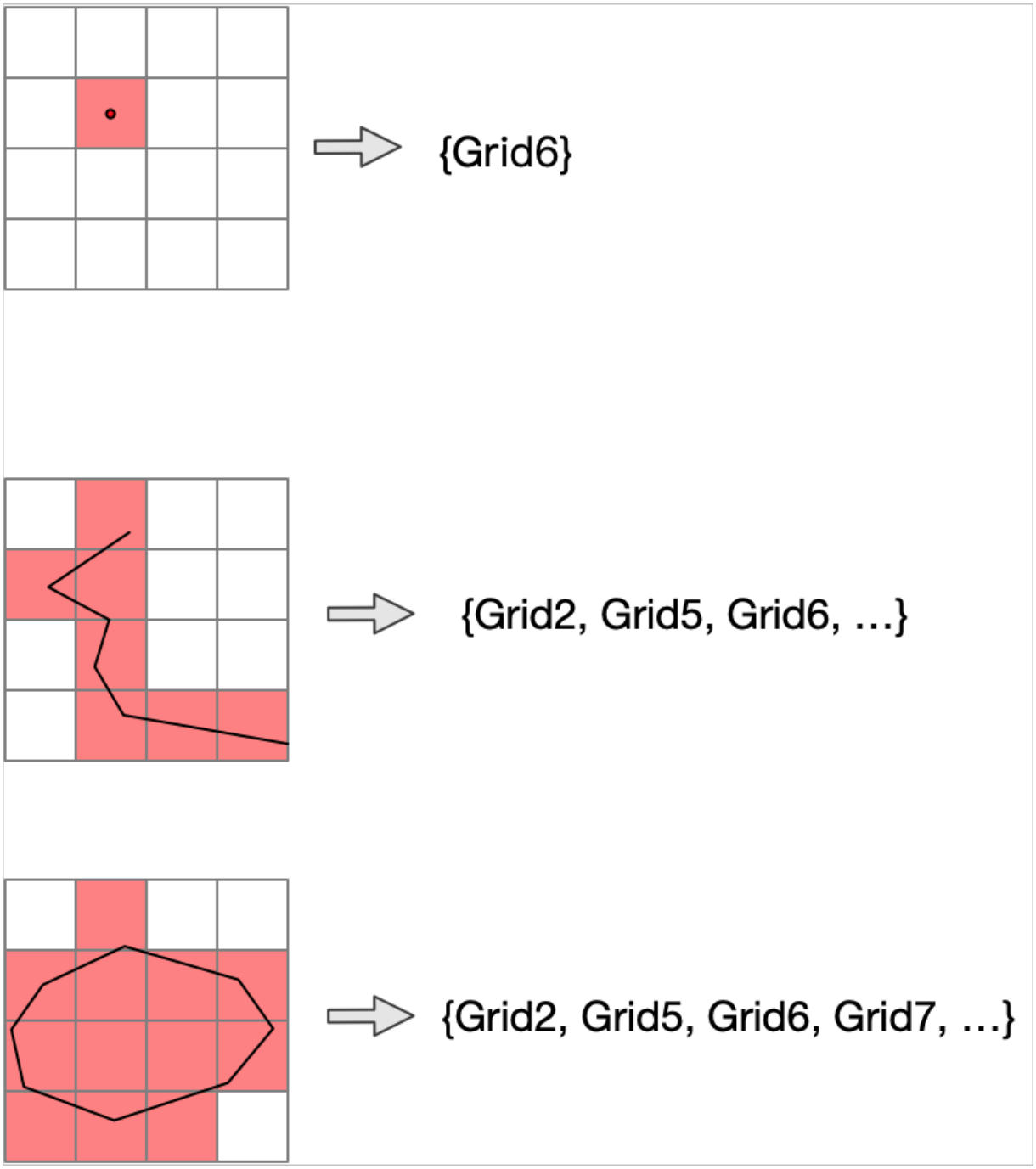
参数

参数名称	描述
geom	需要计算的几何对象。
precision	精度级别，取值：0~31。

描述

几何对象的空间参考必须是CGC2000（SRID=4490），如果不是则会调用空间投影变换函数（ST_Transform）对几何对象的坐标进行转换，请确保能正确转换到CGC2000坐标系统。

本函数将返回几何对象相交的网格对象数组，对于点、线或面数据分别返回对应的网格编码，如下图所示：



示例

```
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}
```

10.Geometry Pyramid SQL参考

10.1. 使用简介

Ganos Geometry Pyramid可以用于二维空间几何大数据的快速显示。

数据要求

- 类型为Point、Line、Polygon、MultiPoint、MultiLine或MultiPolygon的空间数据表。
- 每条记录必须包含一个唯一的正整数类型的ID字段（uint4/uint8）。
- 必须已经为几何字段创建好空间索引。
- 几何数据的坐标系支持任意 `EPSG` 代码的坐标系，输出mvt支持 `EPSG:3857/EPSG:4326` 两种坐标系。

硬件条件

- 1亿条数据建议配置：32核CPU、64 GB内存、500 GB SSD存储空间，可在15分钟左右处理完毕。
- 金字塔的额外占用空间通常不会超过原数据占用空间的1/10。

10.2. 创建

10.2.1. ST_BuildPyramid

为一个空间几何数据表创建矢量金字塔，加速数据显示。

语法

```
boolean ST_BuildPyramid(cstring table, cstring geom, cstring fid, cstring config)
```

参数

参数名称	描述
table	空间几何表名。
geom	几何字段名。
fid	要素标识字段名。
config	创建金字塔的参数，JSON格式的字符串。

config参数说明如下。

参数名称	类型	默认值	说明
name	string	与table名称相同	金字塔的名称。
parallel	int	0	并行构建任务数，需要设置max_prepared_transactions参数，0表示并行最大化。

参数名称	类型	默认值	说明
tileSize	int	1024	瓦片大小，取值需大于0且小于4096。
tileExtend	int	8	瓦片的外扩大小，取值需大于0。
userExtent	array[double]	null	自定义地理轮廓，设置 []， 分别为：minx、miny、maxx、maxy。
splitSize	int	10000	决定索引节点分裂的最大要素数量。数值越大，金字塔越稀疏。
maxLevel	int	16	金字塔的最大层级，取值为0~20。
sourceSRS	int	-1	源数据的坐标系。如果不设置，则读取元数据的 SRID。
destSRS	int	3857	输出瓦片坐标系参考编码（EPSG），仅支持3857和4326两个值。
buildRules	array[object]	null	构建规则，每个object中包含level和value两个值。
└level	array[int]	无	规则适用的层级数组。
└value	object	无	构建规则值。
└filter	string	无	过滤表达式，PostgreSQL的查询语法。
└attrFields	array[string]	无	mvt中的属性字段名称。
└merge	array[string]	无	对数据进行分组合并的过滤条件。

config示例：

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

示例

```
--为空间几何表roads创建矢量金字塔。
select ST_BuildPyramid('roads', 'geom', 'id', '');
st_buildpyramid
----------
t
```

10.3. 删除

10.3.1. ST_DeletePyramid

删除矢量金字塔。

语法

```
boolean ST_DeletePyramid(cstring name)
```

参数

参数名称	描述
name	金字塔的名称。

示例

```
--删除矢量数据的金字塔。
select ST_DeletePyramid('roads');
st_deletepyramid
----------
t
```

10.4. 访问

10.4.1. ST_Tile

从金字塔中按瓦片编号生成一个mvt标准的二进制数据。

语法

```
bytea ST_Tile(cstring name, cstring key);
bytea ST_Tile(cstring name, int x, int y, int z);
```

参数

参数名称	描述
name	金字塔名称。
key	瓦片编号。
x	瓦片编号里的x值。
y	瓦片编号里的y值。
z	瓦片编号里的z值。

描述

key的编号方式为 `'z_x_y'` ，坐标系为 `EPSG:4326` 或 `EPSG:3857` ，返回mvt标准的二进制数据。

> **说明** 当坐标系为 `EPSG:4326` 时，瓦片在x方向的分块数量是y方向分块数量的2倍。最小层级从1开始，且只有 `1_0_0` 或 `1_1_0` 瓦片。

示例

```
select ST_Tile('roads', '3_1_6');
st_tile
----------
0xFFAABB8D8A6678...
select ST_Tile('roads', 1, 6, 3);
st_tile
----------
0xFFAABB8D8A6678...
```

10.4.2. ST_AsPng

从金字塔中按瓦片编号生成一个PNG图片。

语法

```
bytea ST_AsPng(cstring name, cstring key, cstring style);
bytea ST_AsPng(cstring name, int x, int y, int z, cstring style);
```

参数

参数名称	描述
name	金字塔名称。
key	瓦片编号。
x	瓦片编号里的x值。
y	瓦片编号里的y值。
z	瓦片编号里的z值。
style	渲染样式描述参数。格式为JSON的字符串。

style参数说明如下。

参数名称	类型	默认值	说明
background	string	#FFFFFFFF	背景颜色，RGBA颜色值。默认为白色。
line_color	string	#000000FF	点颜色和边线颜色，RGBA颜色值。默认为黑色。
fill_color	string	#F4A460FF	瓦填充颜色，RGBA颜色值。默认为棕色。
line_width	int	1	边线的宽度，单位：像素。
point_size	int	10	点的大小，单位：像素。默认是直径为10像素的圆形。
parallel_unit	int	50000	每个并行任务渲染的要素数量。

style示例如下：

```
{
 "background": "#FFFFFFFF",
 "line_color": "#000000FF",
 "fill_color": "#F4A460FF",
 "line_width": 1,
 "point_size": 10
}
```

描述

key的编号方式为 `'z_x_y'` ，坐标系为 `EPSG:4326` 或 `EPSG:3857` 。

> **说明**
>
> - 当坐标系为 `EPSG:4326` 时，瓦片在x方向的分块数量是y方向分块数量的2倍。最小层级从1开始，且只有 `1_0_0` 或 `1_1_0` 瓦片。
> - 按照渲染样式style返回的PNG图片，图片大小和金字塔的tileSize保持一致。style为空时，使用默认值。

示例

```
select ST_AsPng('roads', '3_1_6', '');
st_aspng
----------
0xFFAABB8D8A6678...
select ST_AsPng('roads', 1, 6, 3, '');
st_aspng
----------
0xFFAABB8D8A6678...
```

11.FDW SQL参考

11.1. 快速入门

外部数据包装器FDW（FOREIGN DATA WRAPPER）是PostgreSQL提供用于访问外部数据的插件，外部数据源包括本实例中其它库中的数据或其他实例的数据。Ganos FDW提供了对于多种空间数据类型的统一访问，会自动将几何空间数据类型映射为Geometry字段类型，从而允许与数据库内部表进行统一地访问与查询。

操作步骤

1. 创建时空数据库FDW插件。

   您可以通过以下两种方式创建扩展。

   - 直接创建Ganos_FDW扩展。

     ```
     CREATE EXTENSION ganos_spatialref;
     CREATE EXTENSION ganos_geometry;
     CREATE EXTENSION ganos_fdw;
     ```

   - 使用CASCADE关键字创建扩展。

     ```
     CREATE EXTENSION Ganos_FDW CASCADE;
     ```

2. 将空间数据文件注册为外表。

   i. 注册Shapefile。

   ```
   SELECT ST_RegForeignTables('OSS://<ak_id>:<ak_secret>@<endpoint>/path/poly.shp');
   ```

   > **说明** ak_id和ak_secret分别为OSS访问的AccessKey信息，具体请参见获取AccessKey ID和Secret。Endpoint为oss的地域节点。为保证数据的可访问性，请确保云数据库与OSS所在Region相同，并使用内部endpoint地址。 相关信息请参考OSS endpoint。

   ii. 通过 `information_schema.foreign_tables` 视图查询注册的FDW表。

   ```
   SELECT foreign_table_name FROM information_schema.foreign_tables ORDER BY foreign_table_name ASC;
   ```

3. 查询外表。

   ```
   SELECT fid, ST_AsText(geom), name, age, height FROM poly WHERE fid = 1;
   ------------
     1 | POLYGON((5 0,0 0,0 10,5 10,5 0)) | ZhangShan |  35 |  1.84
   ```

4. 导入到数据库表中。

   - 如果未创建表，使用如下命令创建并插入数据。

     ```
     CREATE TABLE poly_db AS SELECT * FROM poly;
     ```

   - 如果表已创建，可以通过以下两种方式导入数据。

     - 通过INSERT SELECT方式：

       ```
       INSERT INTO poly_db SELECT * FROM poly;
       ```

- 通过IMPORT FOREIGN SCHEMA方式：

```
CREATE SCHEMA imp;
IMPORT FOREIGN SCHEMA ganos_fdw
 FROM SERVER ganos_fdw_server
 INTO imp;
```

5. （可选）删除扩展。

```
DROP EXTENSION Ganos_FDW CASCADE;
```

11.2. 创建外表并使用

Ganos_FDW遵循fdw接口规范，因此可以通过SQL方式创建外表，实现对外部数据源的操作。

操作步骤

1. 创建服务端。

   配置语法：

```
CREATE SERVER <server_name>
FOREIGN DATA WRAPPER ganos_fdw
OPTIONS (
  datasource 'OSS://<endpoint>/path/file',
  format '<driver>',
  open_options '<config>=<value>[ <config>=<value>]',
  config_options '<config>=<value>[ <config>=<value>]');
```

   关键参数取值含义如下，CREATE SERVER命令的更多参数解释，请参见CREATE SERVER 官方文档：

参数名称	描述
datasource	数据源，必须指向一个合法的OSS地址， OSS路径地址参见ST_ForeignTables函数。 **说明** 此处的OSS地址不需要配置AccessKey ID和AccessKey Secret。
format	使用的数据源驱动程序，可以通过ST_FDWDrivers函数获得。如果传入空字符串，则表示使用默认的驱动尝试访问。
config_options	环境变量参数选项。
open_options	数据源打开选项。

   配置示例：

```
CREATE SERVER myserver
FOREIGN DATA WRAPPER ganos_fdw
OPTIONS (
 datasource 'OSS://<endpoint>/path/poly.shp',
 format 'ESRI Shapefile',
 open_options 'SHAPE_ENCODING=LATIN1',
 config_options '');
```

2. 创建User Mapping。

   配置语法：

```
CREATE USER MAPPING
FOR <user_name>
SERVER <server_name>
OPTIONS (
 user '<oss_ak_id>',
 password '<oss_ak_secret>');
```

   关键参数取值含义如下，**CREATE USER MAPPING**命令的更多参数解释，请参见CREATE USER MAPPING 官方文档：

参数名称	描述
SERVER	服务端名称，与步骤1中的**CREATE SERVER**创建的名称（server_name）一致。
user	AccessKey ID，具体请参见获取AccessKey ID和Secret。
password	AccessKey Secret。

   配置示例：

```
CREATE USER MAPPING
FOR CURRENT_USER
SERVER myserver
OPTIONS (
 user 'id',
 password 'secret');
```

3. 创建外表。

   配置语法：

```
CREATE FOREIGN TABLE <table_name> (
 column_name data_type
 [, ...]
) SERVER <server_name>
OPTIONS (layer '<layer_name>');
```

   关键参数取值含义如下，**CREATE FOREIGN TABLE**命令的更多参数解释，请参见CREATE FOREIGN TABLE 官方文档：

参数名称	描述
SERVER	服务端名称，与步骤1中的CREATE SERVER创建的名称（server_name）一致。
layer	外表对应的图层名称。

配置示例：

```
CREATE FOREIGN TABLE example_table (
 fid bigint,
 name varchar,
 age varchar,
 value varchar
) SERVER myserver
OPTIONS (layer 'poly');
```

4. 导入表定义。

   配置语法：

```
IMPORT FOREIGN SCHEMA ganos_fdw
  [ { LIMIT TO | EXCEPT } ( table_name [, ...] ) ]
  FROM SERVER <server_name>
  INTO <local_schema>
```

   配置示例：

```
CREATE SCHEMA imp;
IMPORT FOREIGN SCHEMA ganos_fdw
 FROM SERVER myserver
 INTO imp;
```

> 说明
>
> - 远程的SCHEMA名称固定为ganos_fdw。
> - 本地SCHEMA可通过CREATE SCHEMA命令创建。
> - IMPORT FOREIGN SCHEMA命令的更多参数解释，请参见IMPORT FOREIGN SCHEMA官方文档。

11.3. ST_FDWDrivers

获得所有Ganos FDW支持的数据源驱动列表。

语法

```
setof record ST_FDWDrivers(out integer id ,
        out text driver_name ,
        out text open_options );
```

参数

参数名称	描述
id	驱动序号。
driver_name	数据驱动名称。
open_options	数据驱动打开参数。

描述

获得Ganos FDW支持的所有数据源驱动列表。其中open_options可以作为CREATE SERVER的参数。

示例

```
select * from
(select (st_fdwdrivers()).*) table_test
where driver_name='ESRI Shapefile';
 id | driver_name |      open_options
 4 | ESRI Shapefile | <OpenOptionList> <Option name='ENCODING' type='string' description='to override the
encoding interpretation of the DBF with any encoding supported by CPLRecode or to "" to avoid any recod
ing'/> <Option name='DBF_DATE_LAST_UPDATE' type='string' description='Modification date to write in DB
F header with YYYY-MM-DD format'/> <Option name='ADJUST_TYPE' type='boolean' description='Whether t
o read w
hole .dbf to adjust Real->Integer/Integer64 or Integer64->Integer field types if possible' default='NO'/> <Opti
on name='ADJUST_GEOM_TYPE' type='string-select' description='Whether and how to adjust layer geomet
ry type from actual shapes' default='FIRST_SHAPE'>  <Value>NO</Value>  <Value>FIRST_SHAPE</Value>  <
Value>ALL_SHAPES</Value> </Option> <Option name='AUTO_REPACK' type='boolean' description='Wheth
er the s
hapefile should be automatically repacked when needed' default='YES'/> <Option name='DBF_EOF_CHAR' t
ype='boolean' description='Whether to write the 0x1A end-of-file character in DBF files' default='YES'/></Op
en
OptionList>
(1 row)
```

11.4. ST_ForeignTables

查询外部数据源中表的名称。

语法

```
setof record ST_ForeignTables(cstring source,
                    cstring driver default '',
                    out integer id,
                    out cstring table_name);
```

参数

参数名称	描述

参数名称	描述
source	数据源，必须指向一个合法的OSS地址。 OSS地址格式为： `OSS://<ak_id>:<ak_secret>@<endpoint>/path/file` ak_id和ak_secret分别为OSS访问的AccessKey信息，具体请参见获取AccessKey ID和Secret。Endpoint为oss的地域节点。为保证数据的可访问性，请确保云数据库与OSS所在Region相同，并使用内部endpoint地址。 相关信息请参考OSS endpoint。
driver	使用的数据源驱动程序，可以通过ST_FDWDrivers函数获得。如果传入空字符串，则表示使用默认的驱动尝试访问。
id	表名序号。
table_name	表名称。

示例

- 使用默认的驱动

```
SELECT
    table_name
FROM
  (select (ST_ForeignTables('OSS://<ak_id>:<ak_secret>@<endpoint>/data')).*) table_test
ORDER BY table_name::text ASC;
 -----------------
 poi
 road
 county
```

- 使用Esri Shapefile驱动

```
SELECT
  table_name
FROM
  (select (ST_ForeignTables('OSS://<ak_id>:<ak_secret>@<endpoint>/data', 'ESRI Shapefile')).*) table_te
st
ORDER BY table_name::text ASC;
 -----------------
 poi
 road
```

11.5. ST_RegForeignTables

将数据源中的表注册为外表。

语法

```
cstring ST_RegForeignTables(cstring source,
                cstring server_name default '',
                cstring driver default '',
                cstring config_option default '',
                cstring open_option defautl '',
                cstring[] tables default NULL,
                cstring prefix default '' );
```

参数

参数名称	描述
source	数据源，必须指向一个合法的OSS地址。 OSS地址格式为： `OSS://<ak_id>:<ak_secret>@<endpoint>/path/file` ak_id和ak_secret分别为OSS访问的AccessKey信息，具体请参见获取AccessKey ID和Secret。Endpoint为oss的地域节点。为保证数据的可访问性，请确保云数据库与OSS所在Region相同，并使用内部endpoint地址。 相关信息请参考OSS endpoint。
driver	使用的数据源驱动程序，可以通过ST_FDWDrivers函数获得。如果传入空字符串，则表示使用默认的驱动尝试访问。
server_name	自动创建的Foreign Server的名称，默认使用 `ganos_fdw_server` 。
config_option	环境变量参数选项。
open_option	数据源打开选项。
tables	需要注册为外表的表名称，可以通过ST_ForeignTables获得。
prefix	注册的外表前缀。

描述

将数据源中的表注册为外表。可以指定服务器名称、数据源打开信息，可以通过 `information_schema.foreign_tables` 视图获取相关信息。

示例

- 只指定路径

```
SELECT ST_RegForeignTables('OSS://<ak_id>:<ak_secret>@<endpoint>/data');
--------------
 Create server 'ganos_fdw_server' successfully
```

- 指定server_name

```
SELECT ST_RegForeignTables('OSS://<ak_id>:<ak_secret>@<endpoint>/data',
                'my_server');
-------------
 Create server 'my_server' successfully
```

- 指定驱动打开选项

```
SELECT ST_RegForeignTables('OSS://<ak_id>:<ak_secret>@<endpoint>/data',
                'myserver',
                'ESRI Shapefile',
                '',
                'SHAPE_ENCODING=LATIN1');
-------------
 Create server 'myserver' successfully
```

- 指定需要注册的表

```
SELECT ST_RegForeignTables('OSS://<ak_id>:<ak_secret>@<endpoint>/data',
                'myserver',
                'ESRI Shapefile',
                '',
                'SHAPE_ENCODING=LATIN1',
                ARRAY['point', 'roads']::cstring[]);
-------------
 Create server 'myserver' successfully
```

- 指定外表前缀

```
SELECT ST_RegForeignTables('OSS://<ak_id>:<ak_secret>@<endpoint>/data',
                'myserver',
                'ESRI Shapefile',
                '',
                'SHAPE_ENCODING=LATIN1',
                ARRAY['point', 'roads']::cstring[],
                'myprefix');
-------------
 Create server 'myserver' successfully
```

12.地图服务

12.1. GeoServer简介

GeoServer是OpenGIS Web服务器规范的J2EE实现，利用GeoServer可以方便的发布地图数据，允许用户对特征数据进行更新、删除、插入操作，通过GeoServer可以在用户之间迅速共享空间地理信息。

GeoServer兼容WMS和WFS两种OGC规范特性，支持PostgreSQL、Shapefile、ArcSDE、Oracle、VPF、MySQL、MapInfo，并支持上百种投影；同时能够将网络地图输出为jpeg、gif、png 、SVG 、KML等格式。GeoServer能够运行在任何基于J2EE/Servlet容器之上，嵌入MapBuilder支持AJAX的地图客户端OpenLayers，除此之外还包括许多其他的特性。

安装包下载

- 适配GeoServer 2.13.x（Geoserver 2.13.x + GeoTools 19.x）
- 适配GeoServer 2.14.x（Geoserver 2.14.x + GeoTools 20.x）
- 适配GeoServer 2.15.x（Geoserver 2.15.x + GeoTools 21.x）
- 适配GeoServer 2.16.x（Geoserver 2.16.x + GeoTools 22.x）
- 适配GeoServer 2.17.x（Geoserver 2.17.x + GeoTools 23.x）

> **说明** 您需要自行下载geoserver，建议geoserver版本为2.14.2及以上。

使用方法： 将上述libs包解压后放到geoserver/WEB-INFO/lib目录下，之后启动容器。

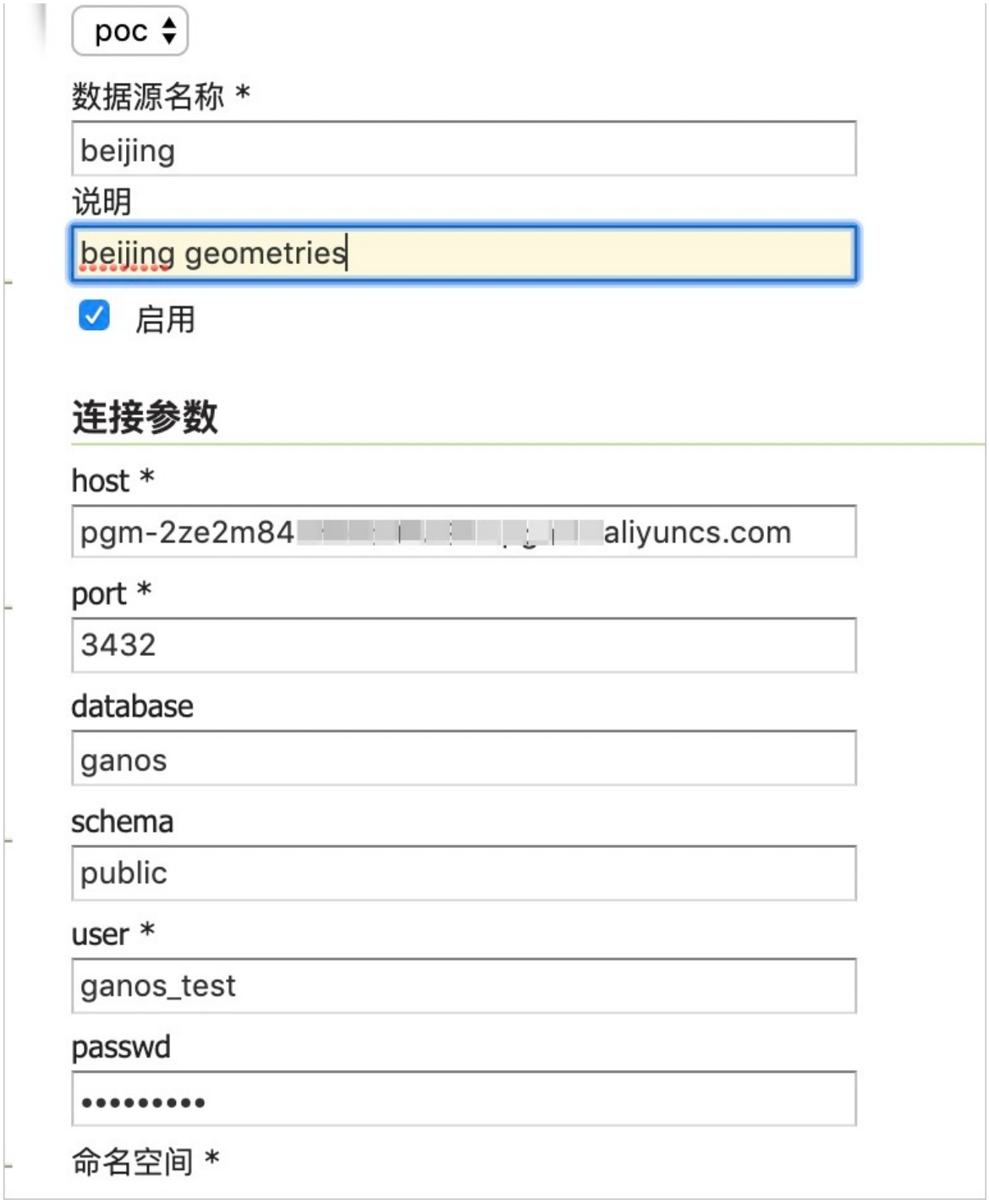
12.2. 发布几何数据

通过GeoServer发布Ganos中的几何数据。

添加数据源

1. 打开GeoServer，选择**数据存储**。
2. 单击**添加新的数据存储**，选择**PostGIS**。

3. 填写Ganos数据库的连接信息。



REST API

几何数据发布后，GeoServer提供给客户调用、访问的REST API，请参见 官网文档

12.3. 发布栅格数据

通过GeoServer发布Ganos中的栅格数据。

添加数据源

1. 打开GeoServer，选择**数据存储**。

2. 单击**添加新的数据存储**，选择**GanosRaster(PG/PolarDB)**。

3. 填写Ganos数据库的连接信息。

参数说明如下。

参数名称	描述	示例
host	数据库地址（IP或RDS连接地址）。	xxxxxxx.pg.rds.aliyuncs.com
port	数据库端口号。	3432
database	数据库名称。	rasterdb
username	数据库用户名。	pguser
password	数据库密码。	123456
schema	表所在的schema。	默认为public

参数名称	描述	示例
table	栅格所在表名。	raster_table
column name	栅格列名称。	raster_column
filter	栅格对象过滤条件，为SQL语句中的where条件。 如果符合条件的栅格有多个，总是使用第一个。	id=1或name='srtm'
name	geoserver显示的栅格名称。	myraster

REST API

- 创建数据源
  - 接口：http://host:port/geoserver/rest/workspaces/{workspace}/coveragestores
  - 方法：POST
  - 参数：
    - workspace：已经创建好的workspace名称。
    - datastore body：示例如下，其中type为固定写法，url中填写阿里云PG/PolarDB的连接参数（json格式）。

```
{
 "coverageStore":{
  "name": "<datasource_name>",
  "type": "GanosRaster(PG/PolarDB)",
  "enabled":"true",
  "workspace":"<wokrspace>",
  "url": "{\"column\":\"<raster_colunm>\",\"database\":\"<database_name>\",\"filter\":\"<raster_filter>\",\"host\":\"<pg_host>\",\"name\":\"<public_name>\",\"password\":\"<user_password>\",\"port\":<pg_port>,\"schema\":\"<schema_name>\",\"ssl\":false,\"table\":\"<raster_table_name>\",\"userName\":\"<user_name>\",\"valid\":true}"
 }
}
```

  - 创建数据源示例

```
{
 "coverageStore":{
  "name": "srtm",
  "type": "GanosRaster(PG/PolarDB)",
  "enabled":"true",
  "workspace":"test",
  "url": "{\"column\":\"rast\",\"database\":\"test_db\",\"filter\":\"name='srtm'\",\"host\":\"pgm-xxxxxxxx.pg.rds.aliyuncs.com\",\"name\":\"srtm_image\",\"password\":\"xxxx\",\"port\":3432,\"schema\":\"public\",\"ssl\":true,\"table\":\"raster_table\",\"userName\":\"raster_user\",\"valid\":true}"
 }
}
```

- 获取数据源
  - 接口：http://host:port/geoserver/rest/workspaces/{workspace}/coveragestores

- 方法：GET
- 参数：
  - workspace：已经创建好的workspace名称。
  - datastore body：示例如下，其中type为固定写法，url中填写阿里云pg/polardb的连接参数（json格式）。

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

- 发布栅格图层
  - 接口：
    http://host:port/geoserver/rest/workspaces/{workspace}/coveragestores/{store}/coverages
  - 方法：POST
  - 参数：
    - workspace：已经创建好的workspace名称。
    - post的body格式如下：

```
{
 "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"
  }
}
```

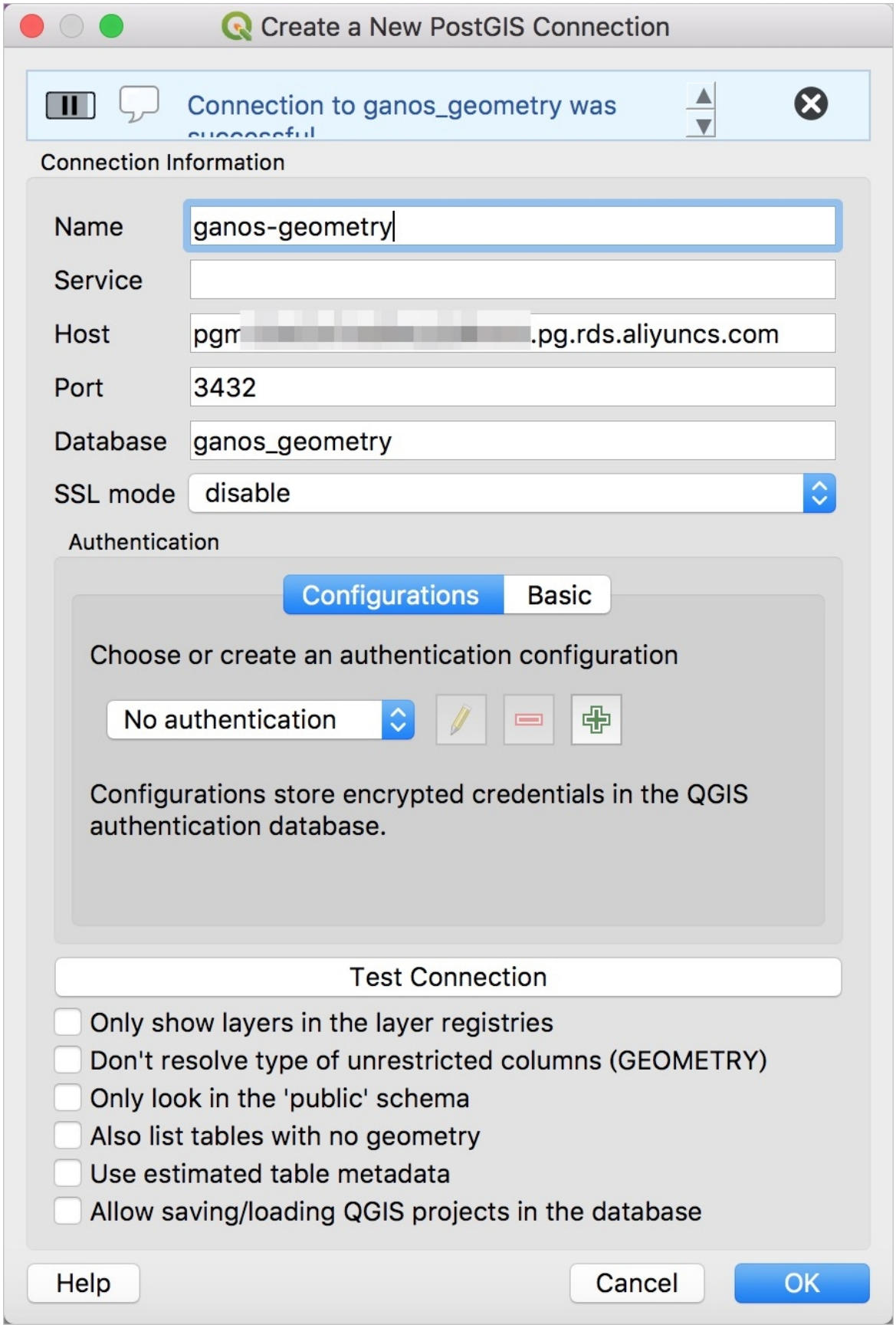
13.桌面应用

13.1. QGIS访问Ganos

QGIS（原称Quantum GIS）是一个用户界面友好的开源桌面端软件，支持多种矢量、栅格格式以及数据库作为数据源，同时支持数据的可视化、管理、编辑、分析以及印刷地图的制作。

创建Ganos连接

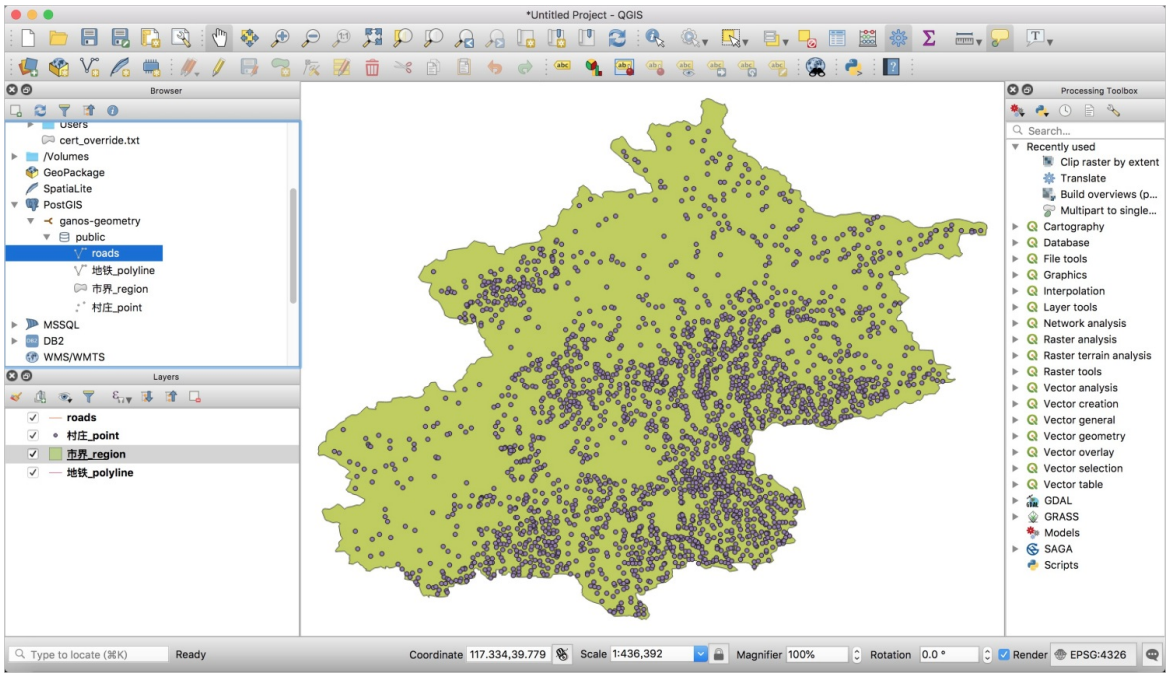
1. 直接创建PostGIS Connection，填入必要项参数，点击**Test Connection**。


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

参数说明如下。

参数	说明
Name	自定义连接名。
Host	数据库实例的IP地址或RDS/PolarDB外网地址，可以从控制台中获得。
Port	数据库实例的端口地址，可以从控制台中获得。
Database	需要连接的数据库名。
SSL Mode	SSL加密状态，选择**Disable**。

2. 找到建立的Ganos数据源，显示出已有的数据库实例以及实例下的空间图层，双击指定的空间图层，即可浏览、查看、编辑Ganos中的空间数据。

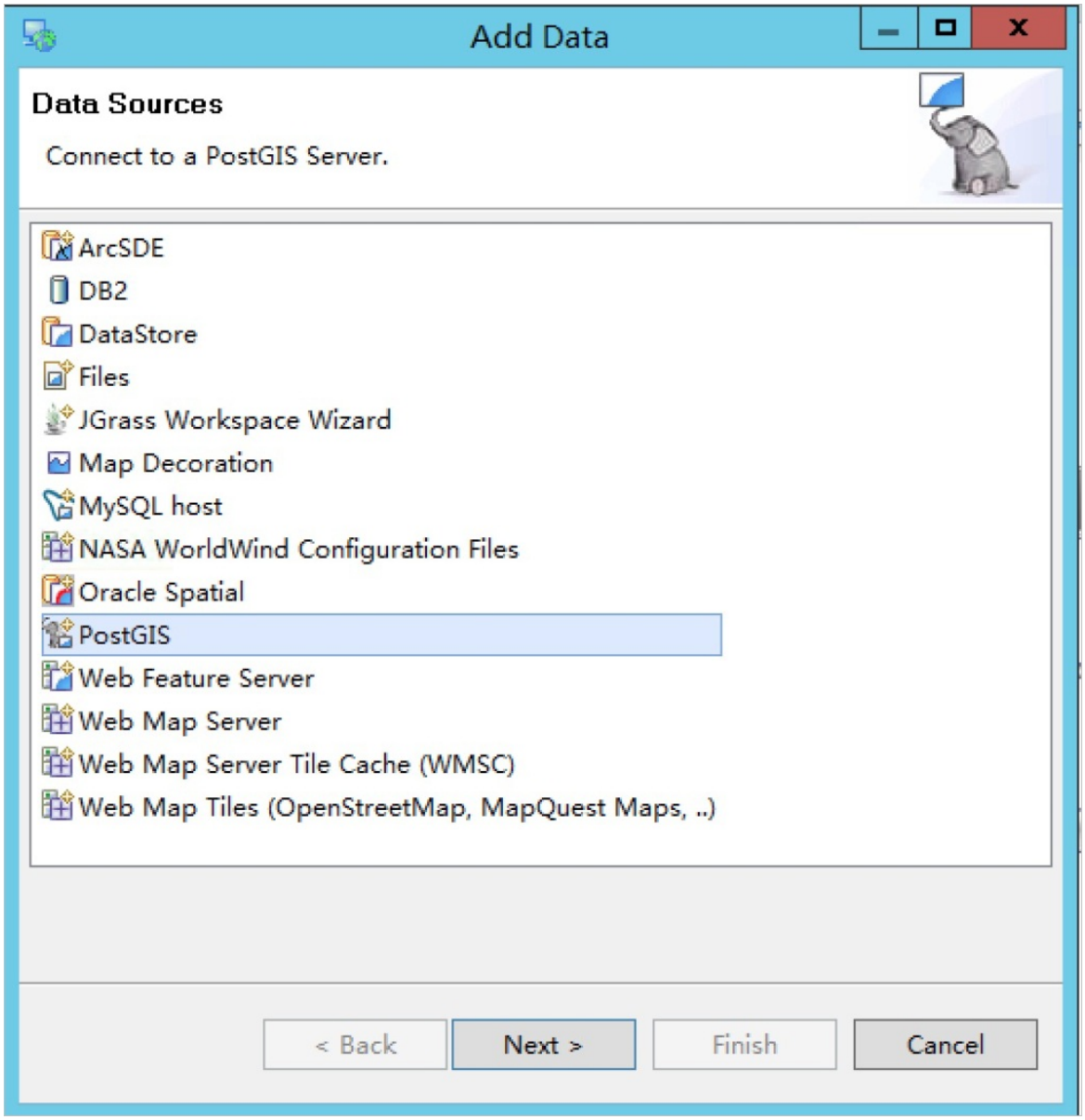


13.2. uDig访问Ganos

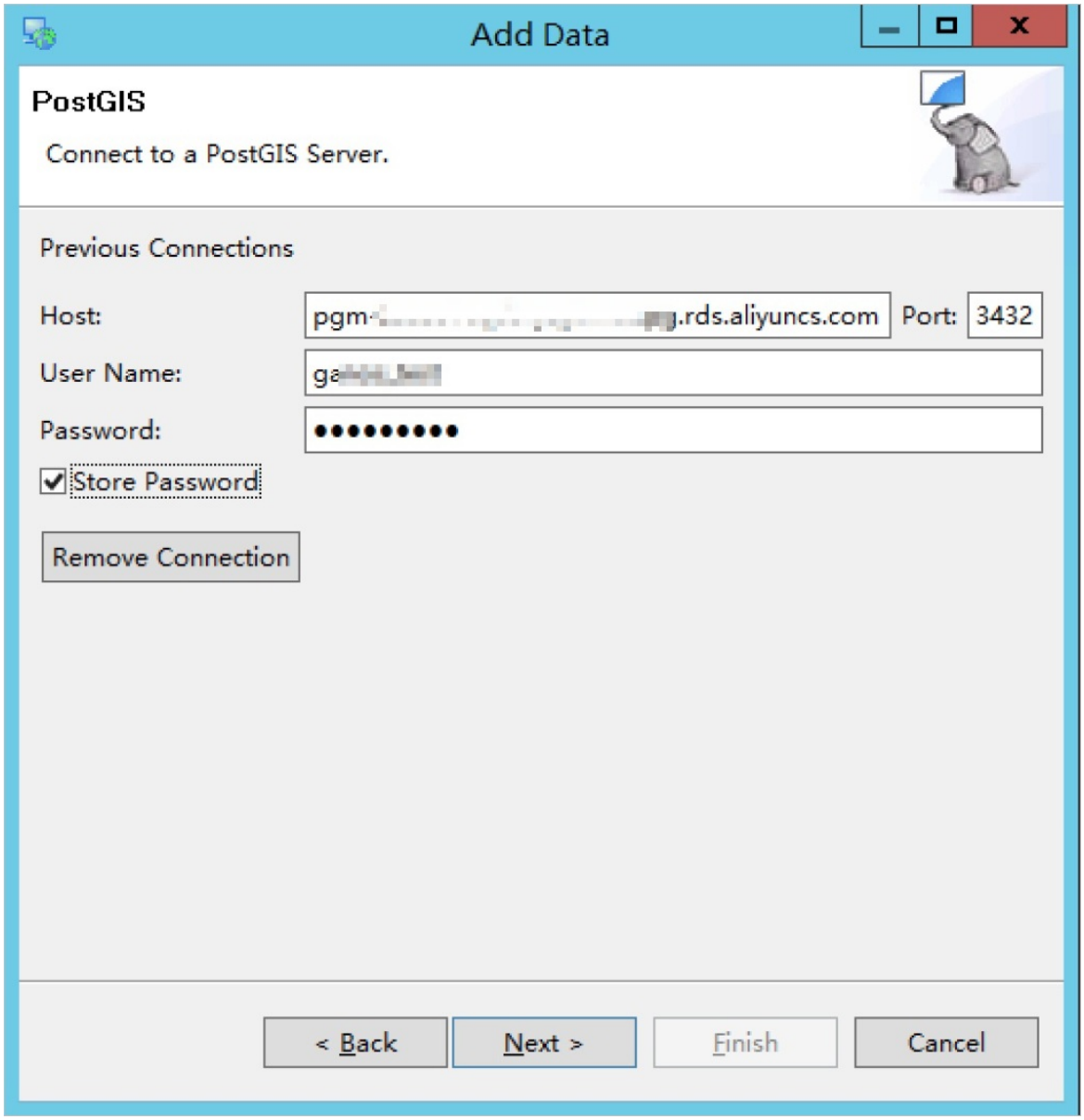
uDig是一款开源的桌面GIS应用和开发框架，可以进行空间数据（如shp地图文件）的编辑和查看，支持OpenGIS标准，对互联网GIS、网络地图服务器和网络功能服务器有特别的加强。uDig提供了一个通用的Java平台来利用开源组件建设空间应用程序。

创建Ganos连接

1. 在软件界面选择**Layer > Add Data**。



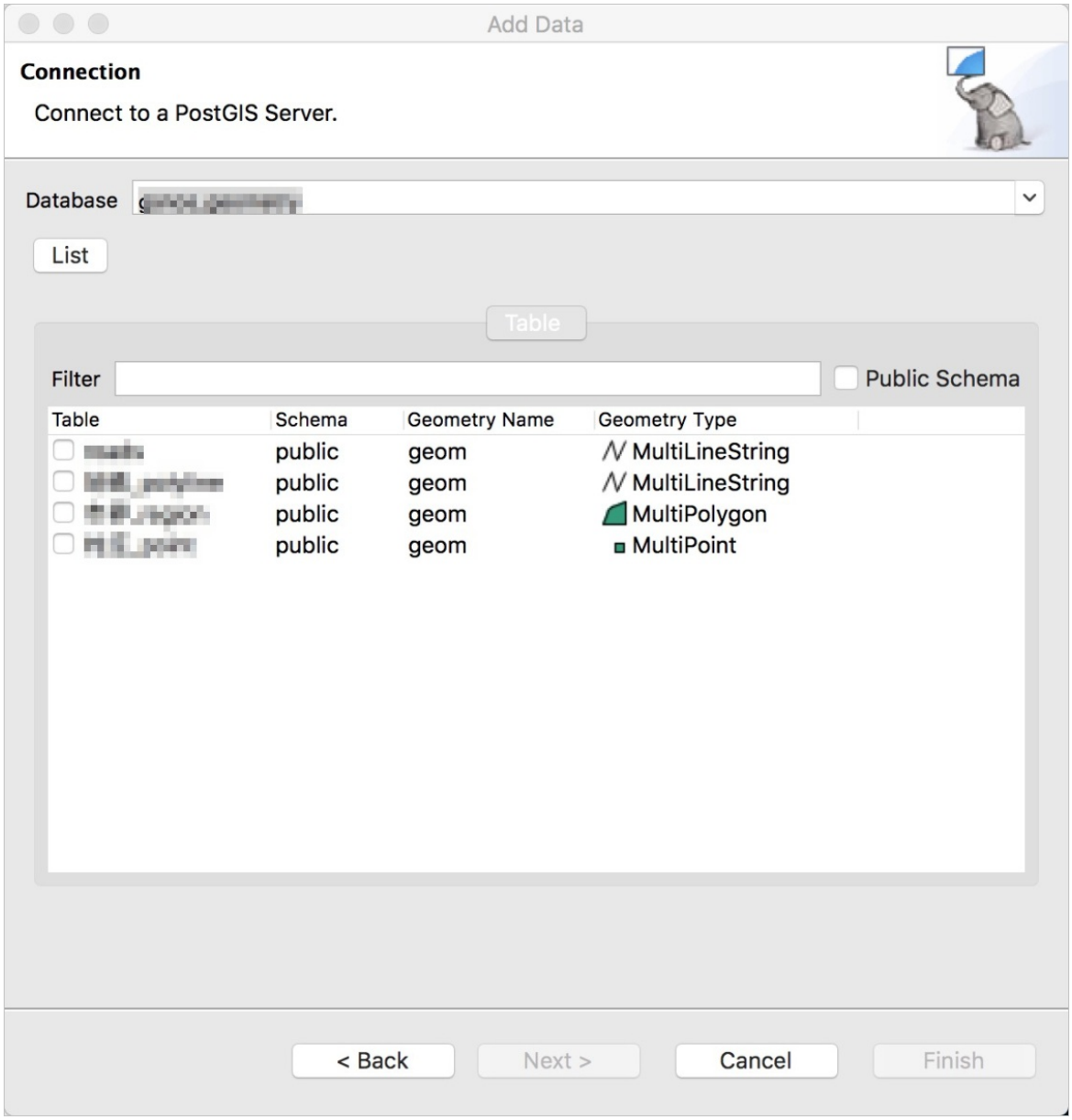
2. 填入Ganos数据库连接信息。



参数说明如下。

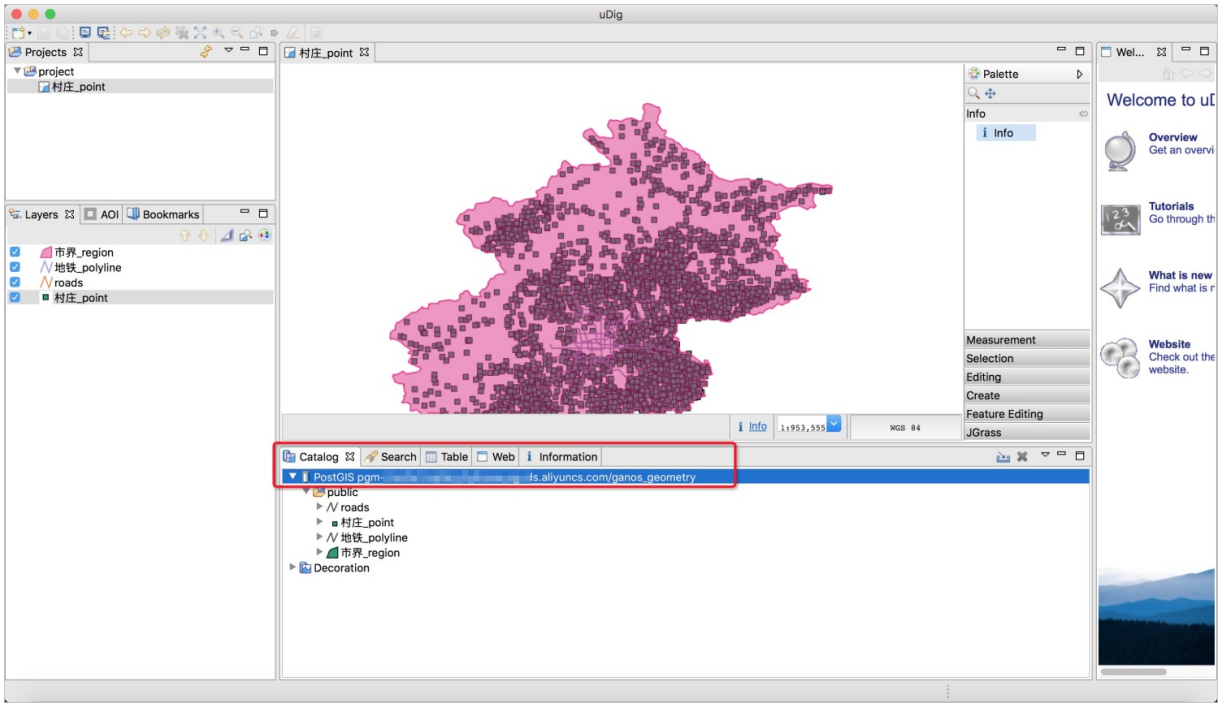
参数	说明
Host	数据库实例的IP地址或RDS/PolarDB外网地址，可以从控制台中获得。
Port	数据库实例的端口地址，可以从控制台中获得。
User Name	数据库用户名。
Password	密码。

3. 选择数据库实例。



数据浏览与操作

连接Ganos后，可以对数据库中对图层进行浏览、查看、编辑Ganos中的空间数据。

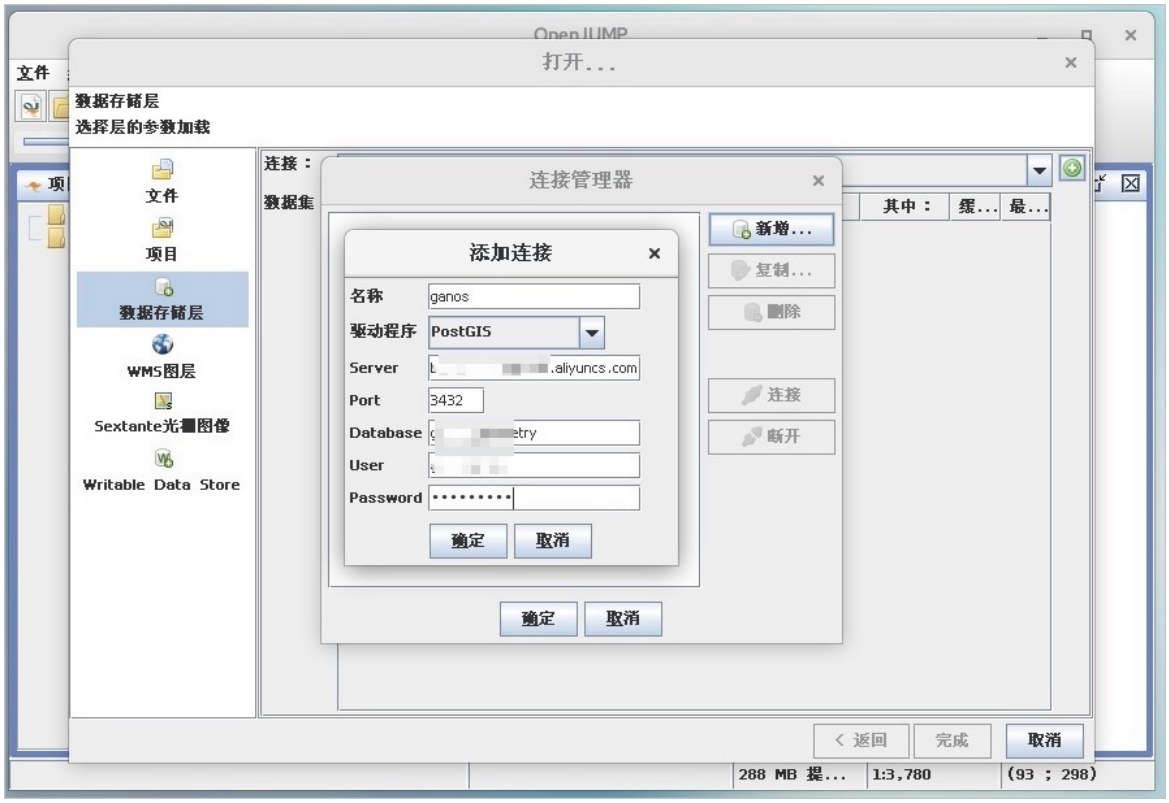


13.3. OpenJump访问Ganos

OpenJUMP是一套基于Java桌面GIS应用程序，其内置了地图可视化功能，可以用来实现特定的空间分析计算，包括Buffer缓冲区分析、Intersection叠加求交、Union叠加求和等空间分析功能，并可以通过插件方式为OpenJUMP进行功能的定制或拓展。

创建Ganos连接

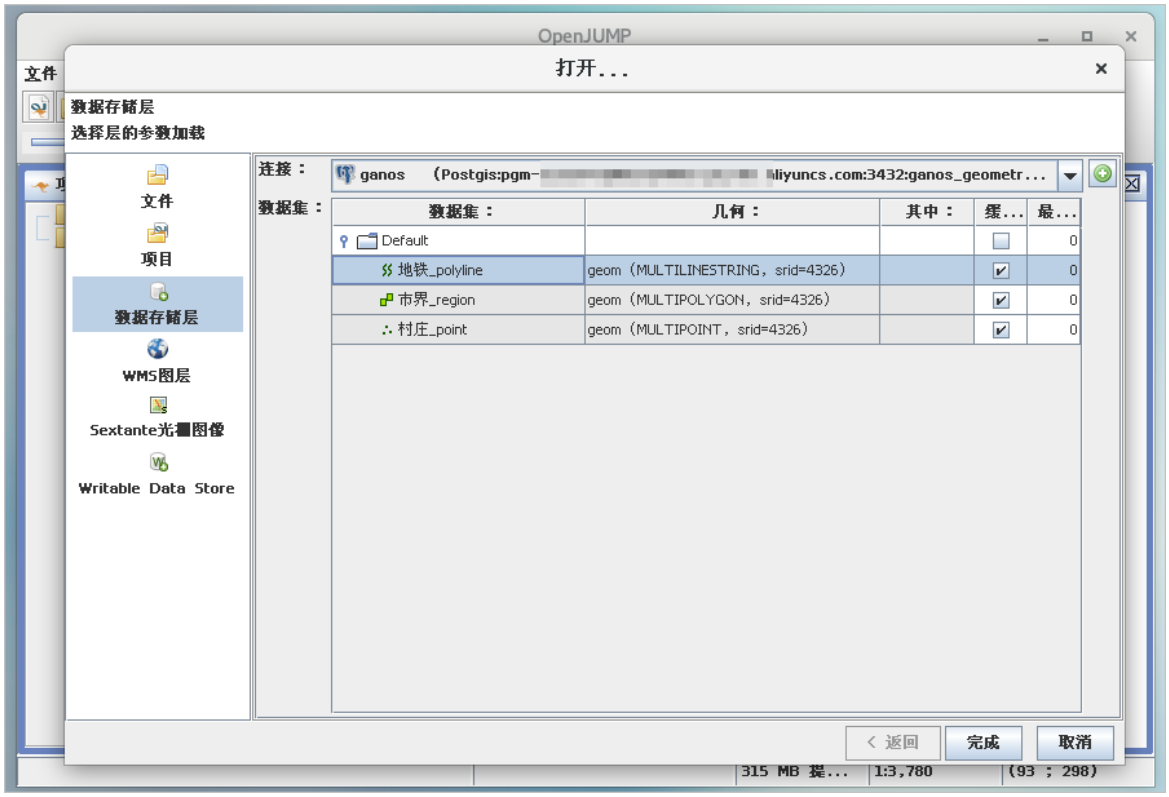
1. 在软件界面选择**文件 > 打开**，选择**数据存储层**。
2. 在**连接管理器**中选择**新增**。
3. 在**添加连接**对话框中填入连接参数。



参数说明如下。

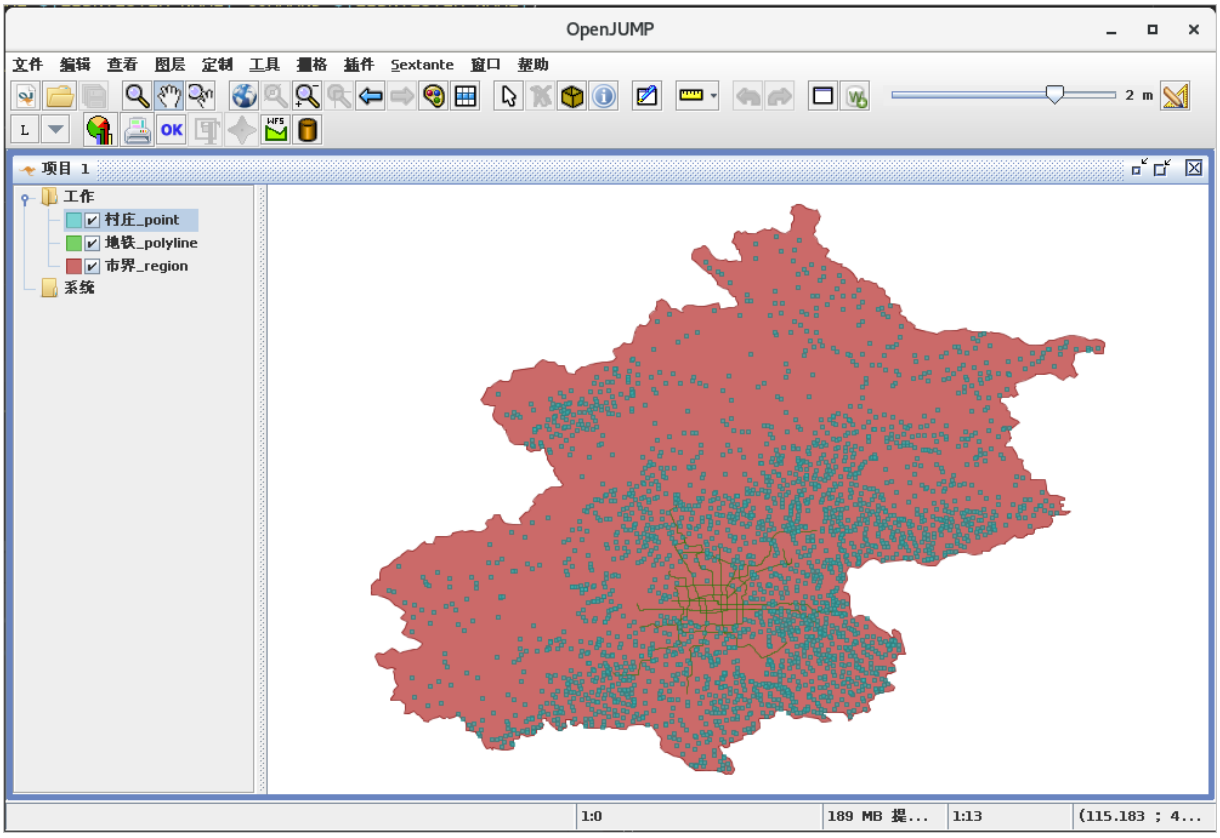
参数	说明
名称	自定义的连接名称。
驱动程序	指定为**PostGIS**。
Server	数据库实例的IP地址或RDS/PolarDB外网地址，可以从控制台中获得。
Port	数据库实例的端口地址，可以从控制台中获得。
Database	需要连接的数据库名。
User	数据库用户名。
Password	密码。

4. 单击**确定**后可以看到ganos库中所有的图层信息。

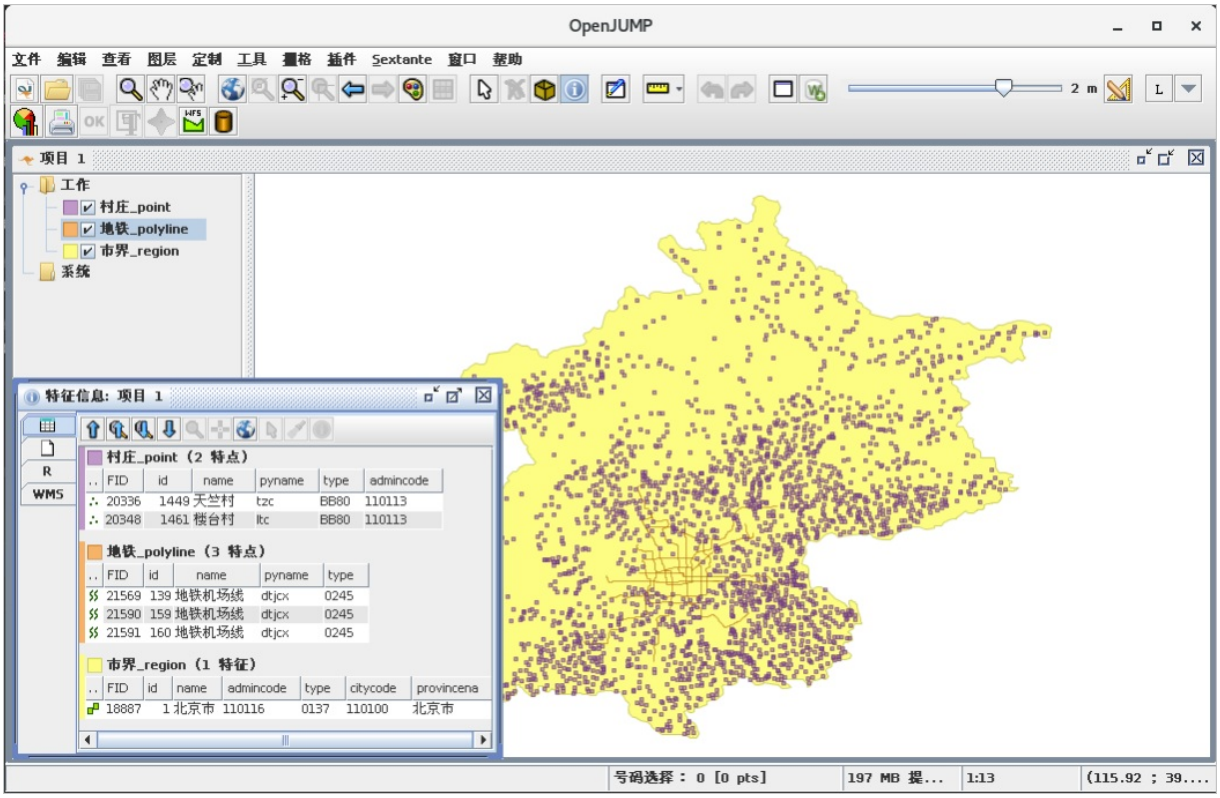


数据浏览与操作

连接后可以对数据进行浏览，可以进行放大、缩小、漫游等基本操作，也可以进行符号化渲染等。



也可以对几何对象进行属性查询。


OpenJUMP
文件 编辑 查看 图层 定制 工具 栅格 插件 Sextante 窗口 帮助
项目 1
工作
村庄_point
地铁_polyline
市界_region
系统
特征信息: 项目 1
R
WMS
村庄_point (2 特点)
FID id name pyname type admincode
20336 1449 天竺村 tzc BB80 110113
20348 1461 楼台村 ltc BB80 110113
地铁_polyline (3 特点)
FID id name pyname type
21569 139 地铁机场线 dtjcx 0245
21590 159 地铁机场线 dtjcx 0245
21591 160 地铁机场线 dtjcx 0245
市界_region (1 特征)
FID id name admincode type citycode provincena
18887 1 北京市 110116 0137 110100 北京市
号码选择： 0 [0 pts]
197 MB 提...
1:13
(115.92 ; 39....

14.常见问题

14.1. 加载栅格数据

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

注意事项

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

ST_ImportFrom函数

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

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

语法

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

示例

1. 创建ganos_raster扩展。

   ```
   CREATE EXTENSION Ganos_Raster CASCADE
   ```

2. 将一个本地文件上的栅格文件加载到Ganos上 。

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

3. 将一个OSS文件上的栅格影像文件加载到Ganos上。

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

14.2. 加载矢量数据

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

准备工作

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

```
CREATE EXTENSION ganos_geometry CASCADE
```

shp2pgsql命令行工具

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

- 语法

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

- 参数说明

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

- 示例

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

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

ogr2ogr命令行工具

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

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

- 语法

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

- 参数说明

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

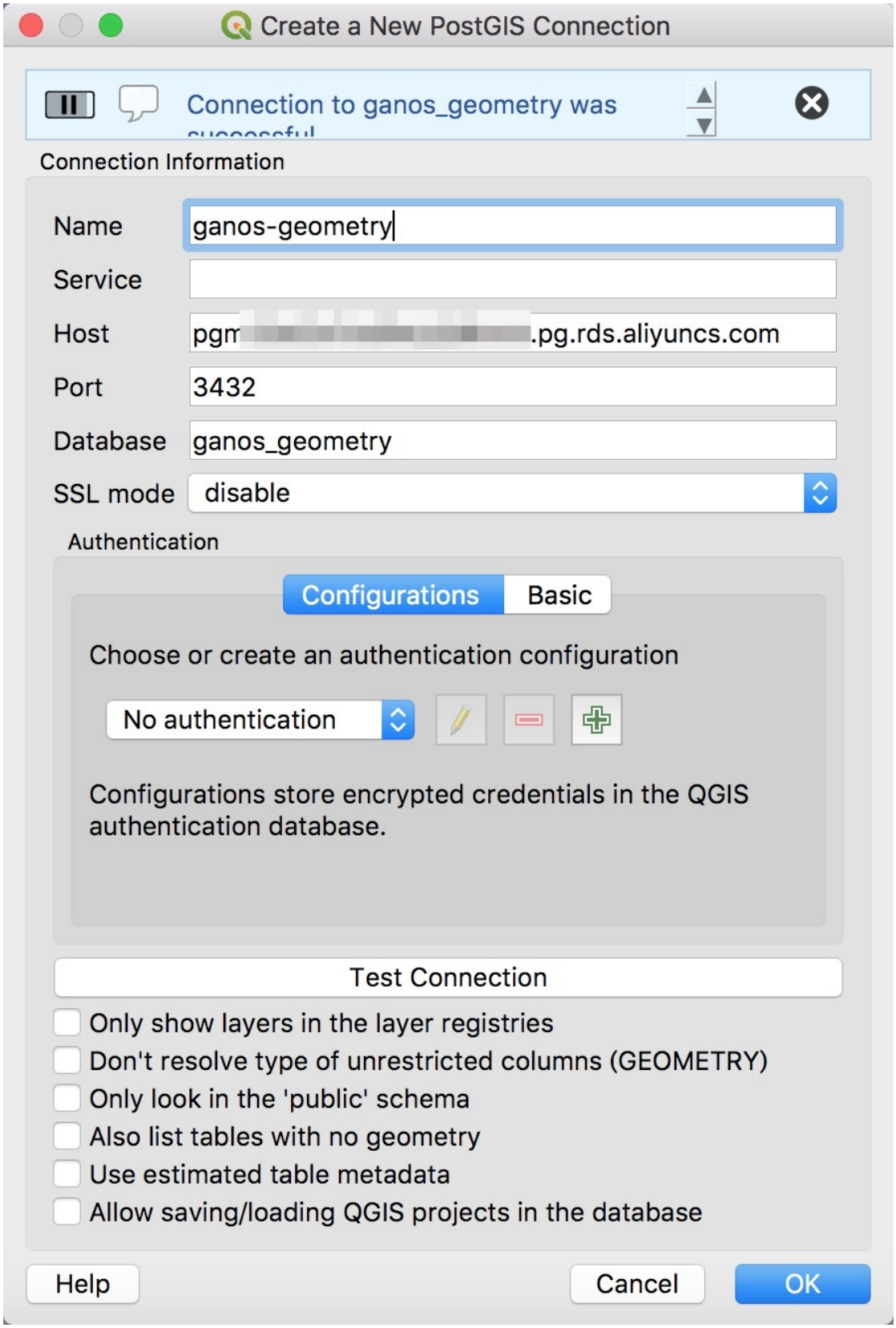
- 示例

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

QGIS桌面工具

QGIS（原称Quantum GIS）是一个用户界面友好的开源桌面端软件，支持多种矢量、栅格格式以及数据库作为数据源，同时支持数据的可视化、管理、编辑、分析以及印刷地图的制作。

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


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

参数说明如下：

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

iii. 单击Test Connection等待连接成功。

2. 加载矢量数据。
   i. 单击菜单栏Database > DB Manager..。
   ii. 在弹出的数据库连接对话框中，选择需要导入的数据源并单击Import Layer/File...。
   iii. 在弹出的导入矢量数据对话框中，指定需要导入的矢量数据并设置导入选项，最后单击OK执行导入操作。

14.3. Trajecotry常见问题

本文为您介绍Trajecotry常见问题和解答，有助于您使用时空数据库。

如何把坐标点数据转换为轨迹对象？

采用ST_MakeTrajectory 构造函数可以将其转换为轨迹对象，设置方法如下：

```
-- 创建扩展
create extension ganos_trajectory cascade;
-- 创建点表
create table points(id integer, x float8, y float8, t timestamp, speed float8);
insert into points values(1, 128.1, 28.1, '2019-01-01 00:00:00', 100);
insert into points values(2, 128.2, 28.2, '2019-01-01 00:00:01', 101);
insert into points values(3, 128.3, 28.3, '2019-01-01 00:00:02', 102);
insert into points values(4, 128.4, 28.4, '2019-01-01 00:00:04', 103);
-- 创建轨迹表
create table traj(id integer, traj trajectory);
-- 插入数据
insert into traj(id, traj)
select 1,
  ST_MakeTrajectory('STPOINT'::leaftype, x, y, 4326, t, ARRAY['speed'], NULL, s, NULL)
FROM (select array_agg(x order by id) as x,
    array_agg(y order by id) as y,
    array_agg(t order by id) as t,
    array_agg(speed order by id) as s
 From points) a;
```

系统自带的ST_MakTrajectory构造函数不满足需求怎么办？

如果系统自带的ST_MakeTrajectory函数参数不符合应用需求，您可以进行自定义扩展。例如轨迹对象包含2个int8、2个float4和1个timestamp类型的属性，可以创建如下构造函数：

```
CREATE OR REPLACE FUNCTION  ST_MakeTrajectory(type leaftype, x float8[], y float8[] ,
        srid integer, timespan timestamp[],attrs_name cstring[], attr1 int8[],
        attr2 int8[], attr3 float4[], attr4 float4[], attr5 timestamp[])
  RETURNS trajectory
  AS '$libdir/libpg-trajectory16','sqltr_traj_make_all_array'
  LANGUAGE 'c' IMMUTABLE Parallel SAFE;
```

其中前6个参数为固定类型的参数，后5个参数为用户自定义的轨迹属性类型。使用时直接使用用户定义的重载函数即可。

如何向轨迹中追加轨迹点？

使用ST_append函数向现有轨迹追加轨迹点，ST_append函数声明如下：

```
trajectory ST_append(trajectory traj, geometry spatial, timestamp[] timespan, text str_theme_json) ;
trajectory ST_append(trajectory traj, trajectroy tail) ;
```

例如，基于点表向轨迹中追加轨迹点方法如下：

```
-- 创建扩展
create extension ganos_trajectory cascade;
-- 创建点表
create table points(id integer, x float8, y float8, t timestamp, speed float8);
insert into points values(1, 128.1, 28.1, '2019-01-01 00:00:00', 100);
insert into points values(2, 128.2, 28.2, '2019-01-01 00:00:01', 101);
insert into points values(3, 128.3, 28.3, '2019-01-01 00:00:02', 102);
insert into points values(4, 128.4, 28.4, '2019-01-01 00:00:04', 103);
-- 创建轨迹表
create table traj(id integer, traj trajectory);
-- 插入数据
insert into traj(id, traj)
select 1,
   ST_MakeTrajectory('STPOINT'::leaftype, x, y, 4326, t, ARRAY['speed'], NULL, s, NULL)
FROM (select array_agg(x order by id) as x,
    array_agg(y order by id) as y,
    array_agg(t order by id) as t,
    array_agg(speed order by id) as s
 From points) a;
-- 插入新轨迹点
insert into points values(5, 128.5, 28.5, '2019-01-01 00:00:05', 105);
insert into points values(6, 128.6, 28.6, '2019-01-01 00:00:06', 106);
insert into points values(7, 128.7, 28.7, '2019-01-01 00:00:07', 107);
-- 基于单点更新轨迹
With point_traj as (
 select ST_MakeTrajectory('STPOINT'::leaftype, x, y, 4326, t, ARRAY['speed'], NULL, s, NULL) AS traj
 FROM (select array_agg(x order by id) as x,
    array_agg(y order by id) as y,
    array_agg(t order by id) as t,
    array_agg(speed order by id) as s
    From points WHERE ID = 5) a
)
```

```
)
Update traj
set traj = ST_append(traj.traj, a.traj)
From point_traj a
WHERE traj.ID=1;
-- 如果使用程序生成SQL也可以写成
With point_traj as (
 select ST_MakeTrajectory('STPOINT'::leaftype, ARRAY[128.5::float8],
    ARRAY[28.5::float8], 4326, ARRAY['2019-01-01 00:00:05'::timestamp],
    ARRAY['speed'], NULL, ARRAY[106::float8], NULL) AS traj
)
Update traj
set traj = ST_append(traj.traj, a.traj)
From point_traj a
WHERE traj.ID =1;
-- 基于多点更新轨迹
With point_traj as (
 select ST_MakeTrajectory('STPOINT'::leaftype, x, y, 4326, t, ARRAY['speed'], NULL, s, NULL) AS traj
 FROM (select array_agg(x order by id) as x,
    array_agg(y order by id) as y,
    array_agg(t order by id) as t,
    array_agg(speed order by id) as s
    From points WHERE ID > 5 ) a
)
Update traj
set traj = ST_append(traj.traj, a.traj)
From point_traj a
WHERE traj.ID =1;
```

如何启用优化压缩功能？

lz4压缩算法是一种优秀的压缩算法，具有更高的压缩率和更快的执行速度。如果要启用lz4压缩算法，设置方法如下：

```
-- 设置使用lz4 压缩
Set toast_compression_use_lz4 = true;
-- 使用pg默认压缩算法
Set toast_compression_use_lz4 = false;
```

如果要对整个数据默认采用lz4压缩算法，设置方法如下：

```
-- 数据库使用lz4压缩
Alter database dbname Set toast_compression_use_lz4 = true;
-- 数据库使用默认压缩
Alter database dbname Set toast_compression_use_lz4 = false;
```

如何设置字符串类型属性默认长度？

guc变量ganos.trajectory.attr_string_length用于设置字符串类型属性。设置方法如下：

```
Set ganos.trajectory.attr_string_length = 32;
```

如何计算某个属性的最大值（最小值/平均值）？

计算某个属性的最大值（最小值/平均值），方法如下：

```
With traj AS (
 select '{"trajectory":{"version":1,"type":"STPOINT","leafcount":2,"start_time":"2010-01-01 11:30:00","end
_time":"2010-01-01 12:30:00","spatial":"SRID=4326;LINESTRING(1 1,3 5)","timeline":["2010-01-01 11:30:00","
2010-01-01 12:30:00"],"attributes":{"leafcount":2,"velocity":{"type":"integer","length":4,"nullable":true,"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;
```

14.4. Ganos插件如何升级

Ganos插件是阿里云自研的时空数据库，提供一系列的数据类型、函数和存储过程，本文介绍Ganos插件的升级方法。

阿里云时空数据库Ganos插件在描述中进行标识。您可以通过如下方法快速找到已安装的Ganos插件并升级。

1. 使用PostgreSQL命令行工具连接数据库。
2. 使用 `\dx` 命令查看插件列表。

   结果示例：

```
                                    List of installed extensions
        Name          | Version |   Schema   |                            Description
--------------------------------------+---------+------------+-------------------------------------------------------------------
---------------------------------------
 address_standardizer          | 2.5.4  | public     | Ganos PostGIS+ address standardizer
 address_standardizer_data_us  | 2.5.4  | public     | Ganos PostGIS+ address standardizer data us
 ganos_address_standardizer    | 4.1    | public     | Used to parse an address into constituent elements.
Generally used to support geocoding address normalization step.
 ganos_address_standardizer_data_us | 4.1    | public     | Address Standardizer US dataset example
 ganos_geometry                | 4.1    | public     | Ganos geometry extension for PostgreSQL
 ganos_geometry_sfcgal         | 4.1    | public     | Ganos geometry SFCGAL functions extension for Postgr
eSQL
 ganos_geometry_topology       | 4.1    | topology   | Ganos geometry topology spatial types and functio
ns extension for PostgreSQL
 ganos_networking              | 4.1    | public     | Ganos networking extension for PostgreSQL
 ganos_spatialref              | 4.1    | public     | Ganos spatial reference extension for PostgreSQL
 ganos_tiger_geocoder          | 4.1    | tiger      | Ganos tiger geocoder and reverse geocoder
 plpgsql                       | 1.0    | pg_catalog | PL/pgSQL procedural language
 postgis                       | 2.5.4  | public     | Ganos PostGIS+
 postgis_sfcgal                | 2.5.4  | public     | Ganos PostGIS+
 postgis_tiger_geocoder        | 2.5.4  | public     | Ganos PostGIS+ tiger geocoder
 postgis_topology              | 2.5.4  | public     | Ganos PostGIS+ topology
```

3. 通过**Description**列的描述信息获取已安装的Ganos插件。

   > **说明** 如果描述中包含Ganos，则表示是Ganos插件。

4. 升级Ganos插件。
   - 如果您的Ganos插件版本大于等于3.1，请使用如下语句对全部Ganos插件进行升级。

     ```
     SELECT ganos_update();
     ```

   - 如果您的Ganos插件版本小于3.1，请参考如下命令，手动创建函数对全部Ganos插件进行升级。

     ```
     CREATE OR REPLACE FUNCTION ganos_update()
       RETURNS text AS
     $$
     DECLARE
       rec RECORD;
       sql text;
     BEGIN
       FOR rec IN
         SELECT extname
         FROM pg_extension
         WHERE extname like 'ganos_%'
         LOOP
           sql = 'ALTER EXTENSION '
             || rec.extname
             || ' UPDATE ';
           RAISE NOTICE '%', sql;
           EXECUTE sql;
         END LOOP;
       return 'All Ganos extensions have updated to latest version';
     END
     $$ LANGUAGE 'plpgsql' volatile STRICT;
     ```

     > **说明** Ganos插件与其他插件的升级方式不同，其他插件升级方式如下：
     >
     > ```
     > ALTER EXTENSION <插件名> UPDATE;
     > ```

15.最佳实践

15.1. Trajectory最佳实践

使用合适的时空索引

合适的索引能加快查询速度，需要根据应用场景的要求来构建合适的索引。可针对轨迹数据类型构建以下索引：

- 空间索引：只针对轨迹的空间范围建立索引，适合只查询轨迹空间范围情况。
- 时间索引：只针对轨迹的时间范围建立索引，适合只查询轨迹时间范围情况。
- 时空复合索引：建立时空联合索引，适合时间和空间同时过滤查询。

```
--创建基于函数的空间索引，加速空间过滤
create index tr_spatial_geometry_index on trajtab using gist (st_trajectoryspatial(traj));
--创建基于函数的时间段索引，加速时间过滤
create index tr_timespan_time_index on trajtab using gist (st_timespan(traj));
--创建基于函数的轨迹起止时间
create index tr_starttime_index on trajtab using btree (st_starttime(traj));
create index tr_endtime_index on trajtab using btree (st_endtime(traj));
--首先创建btree_gist扩展
create extension btree_gist;
--建立btree_gist 起始时间、终止时间、空间复合索引
create index tr_traj_test_stm_etm_sp_index on traj_test using gist (st_starttime(traj),st_endtime(traj),st_t
rajectoryspatial(traj));
```

采用合理的分区表

随着使用时间的增加，数据库中的轨迹数据量也不断增加，导致数据库索引变大，查询变慢。您可考虑采用分区表的模式降低单表数据量。

使用分区表请参见PostgreSQL文档中分区表相关章节。

减少使用字符串类型属性

轨迹属性中如有大量的字符串属性，会导致存储空间浪费和性能下降。

- 如果字符串为固定内容，可以转换为整型进行枚举，建议在程序中进行转换；
- 如果无法避免使用字符串类型属性，可指定字符串类型默认长度，避免空间浪费。

  设置字符串类型默认长度方法如下：

  ```
  -- 设置字符串类型默认长度为32
  Set ganos.trajectory.attr_string_length = 32;
  ```

采用批量轨迹生成模式

使用批量轨迹点生成轨迹模式，避免采取单个轨迹点追加模式。

采用优化压缩模式

lz4 压缩算法是一种优秀的压缩算法，具有更高的压缩率和更快的执行速度。如果要启用lz4压缩算法，设置方法如下：

```
-- 设置使用lz4 压缩
Set toast_compression_use_lz4 = true;
-- 使用pg默认压缩算法
Set toast_compression_use_lz4 = false;
```

如果要对整个数据默认采用lz4压缩算法，设置方法如下：

```
-- 数据库使用lz4压缩
Alter database dbname Set toast_compression_use_lz4 = true;
-- 数据库使用默认压缩
Alter database dbname Set toast_compression_use_lz4 = false;
```