PostGIS 2.2.0dev 开发手册

370 / 686

**Chapter 9**

**栅格参考**

本章下面给出的函数是PostGIS 栅格功能用户可能会使用到的函数，也是目前PostGIS栅格功能所支持的。也有一些一般用户用不到的支持栅格对象的函数。

raster 类型是一种新的PostGIS数据类型，用于存储和分析栅格数据

关于怎样从栅格文件中load 栅格数据到数据库中，参考5.1节

本章的一些关于栅格的样例都基于下面的一个栅格表，表结构如下：

CREATE TABLE dummy\_rast(rid integer, rast raster);

INSERT INTO dummy\_rast(rid, rast)

VALUES (1,

(’01’ -- little endian (uint8 ndr)

||

’0000’ -- version (uint16 0)

||

’0000’ -- nBands (uint16 0)

||

’0000000000000040’ -- scaleX (float64 2)

||

’0000000000000840’ -- scaleY (float64 3)

||

’000000000000E03F’ -- ipX (float64 0.5)

||

’000000000000E03F’ -- ipY (float64 0.5)

||

’0000000000000000’ -- skewX (float64 0)

||

’0000000000000000’ -- skewY (float64 0)

||

’00000000’ -- SRID (int32 0)

||

’0A00’ -- width (uint16 10)

||

’1400’ -- height (uint16 20)

)::raster

),

-- Raster: 5 x 5 pixels, 3 bands, PT\_8BUI pixel type, NODATA = 0

(2, (’01000003009A9999999999A93F9A9999999999A9BF000000E02B274A’ ||

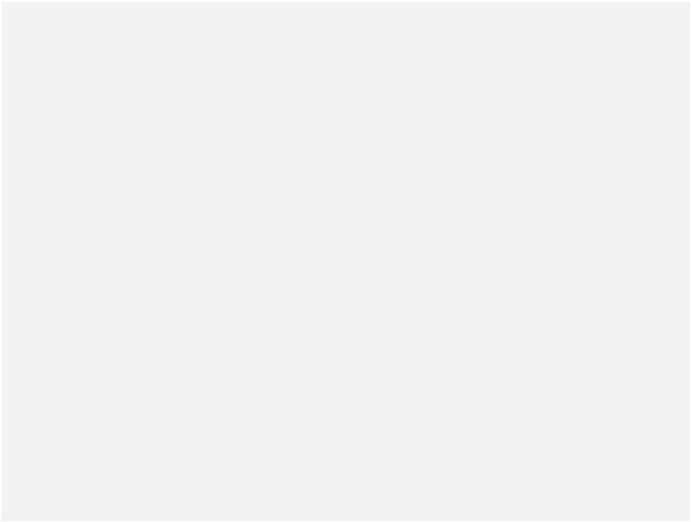
’41000000007719564100000000000000000000000000000000 ←

FFFFFFFF050005000400FDFEFDFEFEFDFEFEFDF9FAFEF’ ||

’ ←

EFCF9FBFDFEFEFDFCFAFEFEFE04004E627AADD16076B4F9FE6370A9F5FE59637AB0E54F58617087040046566487A1506C

’)::raster);



PostGIS 2.2.0dev 开发手册

371 / 686

**9.1**

**9.1.1**

**支持栅格功能的数据类型**

**geomval**

geomval — 一个包含两个字段的复合空间类型——geom字段属性（包含一个几何对象）和val字段属性（包含一个栅格波段的双精度像素值）

**描述**

一个包含两个字段的复合空间类型——geom字段属性（包含一个几何对象）和val字段属性（包含一个栅格波段的双精度像素值）。函数ST\_DumpAsPolygon会使用该类型，栅格交集函数系列的输出类型是该类型，以便将栅格波段展开成基本的几何多边形类型对象。（geometry polygon对象）

**相关请参考**

Section 13.5

**9.1.2**

**addbandarg**

addbandarg — 一个复合数据类型，是函数ST\_AddBand的输入类型，该函数用于定义一个新波段的属性和属性的初始值

**描述**

一个复合数据类型，是函数ST\_AddBand的输入类型，该函数用于定义一个新波段的属性和属性的初始值。

该复合类型包含的波段属性如下：

**index**integer 从1开始的整型值，表示添加到栅格波段中的新波段的位置。如果为空，那么新波段将会追加在栅格波段的末尾。

**pixeltype** text 新波段的像元类型。像元类型的定义参考函数ST\_BandPixelType.

**initialvalue** double precision 新波段的所有像元的初始值

**nodataval** double precision 新波段的值为NODATA，如果为空，新波段将不指定NODATA值。

**相关请参考**

ST\_AddBand

**9.1.3**

**rastbandarg**

rastbandarg —一种复合数据类型，用于标识一个栅格和该栅格上的某个波段

**描述**

一种复合数据类型，用于标识一个栅格和该栅格上的某个波段

**rast** raster类型即输入栅格

**nband**integer类型，下标从1开始，表示栅格的波段位置

PostGIS 2.2.0dev 开发手册

372 / 686

**相关请参考**

ST\_MapAlgebra

**9.1.4**

**raster**

raster — 栅格空间数据类型

**描述**

raster类型是一种空间数据类型，用于表示jpeg,tiff,png,DEM模型这样文件格式的数据。每一个栅格至少有1个波段，每个波段又有一系列像素值。栅格数据是转换成地理坐标的。

**注意**

需要PostGIS在安装时候有GDAL库支持。目前栅格类型数据可以隐式转换成geometry类型，但是该转换回返回该栅格的凸包。这种自动转换将来可能会移除掉，因此不要依赖这个自动转换特性。

**转换方式**

这里列出了该数据类型的自动转换，当然显式转换也行

**相关请参考**

Chapter 9

**9.1.5**

**reclassarg**

reclassarg — 这是一个复合数据类型，用于函数ST\_Reclass，该函数用于定义栅格重分类的方式

**描述**

这是一个复合数据类型，用于函数ST\_Reclass，该函数用于定义栅格重分类的方式

**nband**整型值，要重分类的波段数

**reclassexpr**text类型，范围表达式，有逗号分隔的map\_range\_mappings组成。而map\_range\_mappings是用冒号：分隔的表达式，用于定义旧的波段值和重分类后的新的波段值。

左半括号(的意思是大于，右半括号)意思是小于。左中括号[意思是大于等于，右中括号]意思是小于等于，列出如下：

1. [a-b] = a <= x <= b

2. (a-b] = a < x <= b

3. [a-b) = a <= x < b

4. (a-b) = a < x < b

小括号标记是可以省略的，因此a-b与(a-b)意义是一样的



|  |  |
| --- | --- |
| 目标类型 | 转换方式 |
| geometry | 自动转换 |

PostGIS 2.2.0dev 开发手册

373 / 686

**pixeltype**函数ST\_BandPixelType中所使用的像元类型

**nodataval**被当作 No Data的双精度值。对于支持透明度的图像输出，这些值将为空

译者注：下面样例里面的ROW是PostgreSQL的一个函数，用于构造行

**样例: Reclassify band 2 as an 8BUI where 255 is nodata value**

SELECT ROW(2, ’0-100:1-10, 101-500:11-150,501 - 10000: 151-254’, ’8BUI’, 255)::reclassarg;

**样例: Reclassify band 1 as an 1BB and no nodata value deﬁned**

SELECT ROW(1, ’0-100]:0, (100-255:1’, ’1BB’, NULL)::reclassarg;

**相关请参考**

ST\_Reclass

**9.1.6**

**summarystats**

summarystats — 一个复合类型字段，该类型是ST\_SummaryStats 和 ST\_SummaryStatsAgg 函数返回的类型值

**描述**

一个复合类型字段，该类型是ST\_SummaryStats和ST\_SummaryStatsAgg函数返回的类型值

**count** integer 类型，统计的像元个数

**sum**双精度类型，统计所有像素值的总和

**mean**双精度类型，计算所有像素值的算术平均值

**stddev**双精度类型，计算所有像素值的标准差

**min**双精度类型，计算像素值的最小值

**max**双精度类型，计算像素值的最大值

**相关请参考**

ST\_SummaryStats, ST\_SummaryStatsAgg

**9.1.7**

**unionarg**

unionarg — 一种复合参数类型，函数ST\_Union的输入参数，包含待处理的波段和UNION波段时候的方式

**描述**

一种复合参数类型，函数ST\_Union的输入参数，包含待处理的波段和UNION波段时候的方式

**nband** integer类型，下标从1开始，表示每个输入栅格要处理的波段

**uniontype** text 类型，表示UNION操作的类型，该类型在函数ST\_Union中有定义说明

PostGIS 2.2.0dev 开发手册

374 / 686

**相关请参考**

ST\_Union

**9.2**

**9.2.1**

**栅格管理函数**

**AddRasterConstraints**

AddRasterConstraints — 给一个栅格表的指定列添加一个栅格约束，约束信息包括参考系，缩放比例，块大小，对齐方式，波段，波段类型和一个标志栅格列是否是被锁住的信息。该表必须有约束条件可约束的数据。如果添加约束成功，则返回true，否则提升报错

**用法**

boolean AddRasterConstraints(name rasttable, name rastcolumn, boolean srid, boolean scale\_x, boolean scale\_y, boolean

blocksize\_x, boolean blocksize\_y, boolean same\_alignment, boolean regular\_blocking, boolean num\_bands=true , boolean

pixel\_types=true , boolean nodata\_values=true , boolean out\_db=true , boolean extent=true );

boolean AddRasterConstraints(name rasttable, name rastcolumn, text[] VARIADIC constraints);

boolean AddRasterConstraints(name rastschema, name rasttable, name rastcolumn, text[] VARIADIC constraints);

boolean AddRasterConstraints(name rastschema, name rasttable, name rastcolumn, boolean srid=true, boolean scale\_x=true,

boolean scale\_y=true, boolean blocksize\_x=true, boolean blocksize\_y=true, boolean same\_alignment=true, boolean regular\_blocking=fa

boolean num\_bands=true, boolean pixel\_types=true, boolean nodata\_values=true , boolean out\_db=true , boolean extent=true );

**描述**

为在栅格列信息表raster\_columns中的列加上约束条件。参数rastschema 是表的schema名。SRID值必须是SPATIAL\_REF\_SYS 表中存在的。.

raster2pgsql 该加载工具使用该函数来注册栅格表

必须传递有效的约束名，参考5.2.1获取更多信息

· blocksize 同时设置X和Y的blocksize

· blocksize\_x 设置每个瓦片的X值（即瓦片宽度）译者注：关于瓦片，请参考(http://baike.baidu.com/link?url=A8Vw5XmWEekSrNGIZ5zsHRMDOMGI3q23XrrL8rJhxs813oREZdJeiNy7HZSTamlWWd1j3xBHzEYy8e2pABRgwa)

· blocksize\_y 设置每个瓦片的Y值（即高度）

· extent 计算整个表的几何对象的边界，并且对所有栅格添加约束，使其在几何对象的边界内。

· num\_bands 波段数

· pixel\_types读取每个段的像元类型，确保所有的段有相同的像元类型。

· regular\_blocking 让栅格空间上唯一 (没有两个栅格可以空间上相同) ，并添加瓦片coverage对齐约束 (栅格以coverage方式对齐)

· same\_alignment 确保任意两个瓦片都有相同的对齐方式。意味着，任意两个瓦片比较都返回true参考ST\_SameAlignment

· srid ：确保所有栅格数据有SRID值

· More –上述所列函数的其他参数

PostGIS 2.2.0dev 开发手册

375 / 686

**注意**

这个函数会引用到表中已有数据的约束。因此，你需要先创建栅格列，然后加载数据到该表中。

**注意**

在你已经创建了约束情况下，但却需要加载更多的数据到表中，你可能需要先运行函数DropRasterConstraints来删除约束，如果你的数据的对应的几何对象的边界已经改变了的话

可用版本：2.0.0

**样例: Apply all possible constraints on column based on data**

CREATE TABLE myrasters(rid SERIAL primary key, rast raster);

INSERT INTO myrasters(rast)

SELECT ST\_AddBand(ST\_MakeEmptyRaster(1000, 1000, 0.3, -0.3, 2, 2, 0, 0,4326), 1, ’8BSI’:: ←

text, -129, NULL);

SELECT AddRasterConstraints(’myrasters’::name, ’rast’::name);

-- verify if registered correctly in the raster\_columns view --

SELECT srid, scale\_x, scale\_y, blocksize\_x, blocksize\_y, num\_bands, pixel\_types,

nodata\_values

FROM raster\_columns

WHERE r\_table\_name = ’myrasters’;

srid | scale\_x | scale\_y | blocksize\_x | blocksize\_y | num\_bands | pixel\_types|

←

←

nodata\_values

------+---------+---------+-------------+-------------+-----------+-------------+-------------- - ←

4326 |

2 |

2 |

1000 |

1000 |

1 | {8BSI}

| {0}

**样例: Apply single constraint**

CREATE TABLE public.myrasters2(rid SERIAL primary key, rast raster);

INSERT INTO myrasters2(rast)

SELECT ST\_AddBand(ST\_MakeEmptyRaster(1000, 1000, 0.3, -0.3, 2, 2, 0, 0,4326), 1, ’8BSI’:: ←

text, -129, NULL);

SELECT AddRasterConstraints(’public’::name, ’myrasters2’::name, ’rast’::name,’ ←

regular\_blocking’, ’blocksize’);

-- get notice--

NOTICE:

NOTICE:

NOTICE:

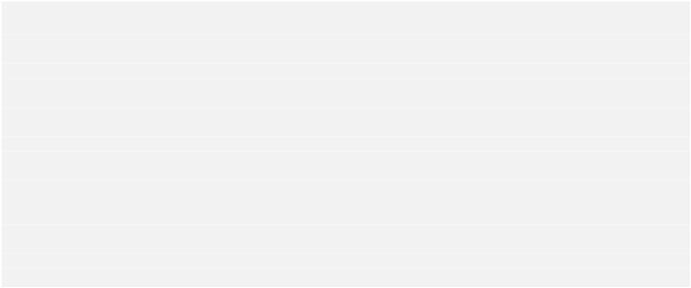
Adding regular blocking constraint

Adding blocksize-X constraint

Adding blocksize-Y constraint

**相关请参考**

Section 5.2.1, ST\_AddBand, ST\_MakeEmptyRaster, DropRasterConstraints, ST\_BandPixelType, ST\_SRID



PostGIS 2.2.0dev 开发手册

376 / 686

**9.2.2**

**DropRasterConstraints**

DropRasterConstraints — 删除PostGIS的栅格表的列上的栅格约束。如果你需要重装载或更新栅格列的数据，该函数很有用

**用法**

boolean DropRasterConstraints(name rasttable, name rastcolumn, boolean srid, boolean scale\_x, boolean scale\_y, boolean

blocksize\_x, boolean blocksize\_y, boolean same\_alignment, boolean regular\_blocking, boolean num\_bands=true, boolean pixel\_types=t

boolean nodata\_values=true, boolean out\_db=true , boolean extent=true);

boolean DropRasterConstraints(name rastschema, name rasttable, name rastcolumn, boolean srid=true, boolean scale\_x=true,

boolean scale\_y=true, boolean blocksize\_x=true, boolean blocksize\_y=true, boolean same\_alignment=true, boolean regular\_blocking=fa

boolean num\_bands=true, boolean pixel\_types=true, boolean nodata\_values=true, boolean out\_db=true , boolean extent=true);

boolean DropRasterConstraints(name rastschema, name rasttable, name rastcolumn, text[] constraints);

**描述**

删除PostGIS的栅格表的列上的栅格约束，栅格约束由函数AddRasterConstraints添加。如果你需要重装载或更新栅格列的数据，该函数很有用。如果你想删除栅格表或栅格类型的列，不需要用这个函数（译者注：因为会自动把栅格约束删除掉）

删除栅格表使用标准DDL语句

DROP TABLE mytable

删除一个表的栅格列，使用标准SQL

ALTER TABLE mytable DROP COLUMN rast

如果带有栅格类型的表或列被删除了，该表将会从记录栅格列的raster\_columns 表中删除。然而如果只是栅格约束删除了，那么该表还会列出栅格列的记录，但除了栅格列名和表名外，没有其他信息

可用版本：2.0.0

**样例**

SELECT DropRasterConstraints (’myrasters’,’rast’);

----RESULT output ---

t

-- verify change in raster\_columns --

SELECT srid, scale\_x, scale\_y, blocksize\_x, blocksize\_y, num\_bands, pixel\_types,

nodata\_values

FROM raster\_columns

WHERE r\_table\_name = ’myrasters’;

srid | scale\_x | scale\_y | blocksize\_x | blocksize\_y | num\_bands | pixel\_types|

←

←

nodata\_values

------+---------+---------+-------------+-------------+-----------+-------------+-------------- - ←

0 |

|

|

|

|

|

|

**相关请参考**

AddRasterConstraints

PostGIS 2.2.0dev 开发手册

377 / 686

**9.2.3**

**PostGIS\_GDAL\_Version**

PostGIS\_GDAL\_Version —返回PostGIS使用的GDAL库的版本

**用法**

text PostGIS\_GDAL\_Version();

**描述**

返回PostGIS使用的GDAL库的版本。如果找到了该库的数据文件，该函数还会检测并上报相关信息

**样例**

SELECT PostGIS\_GDAL\_Version();

postgis\_gdal\_version

-----------------------------------

GDAL 1.11dev, released 2013/04/13

**相关请参考**

postgis.gdal\_datapath

**9.2.4**

**PostGIS\_Raster\_Lib\_Build\_Date**

PostGIS\_Raster\_Lib\_Build\_Date —上报栅格库的完整构建日期

**用法**

text PostGIS\_Raster\_Lib\_Build\_Date();

**描述**

上报栅格库的完整构建日期

**样例**

SELECT PostGIS\_Raster\_Lib\_Build\_Date();

postgis\_raster\_lib\_build\_date

-----------------------------

2010-04-28 21:15:10

**相关请参考**

PostGIS\_Raster\_Lib\_Version

**9.2.5**

**PostGIS\_Raster\_Lib\_Version**

PostGIS\_Raster\_Lib\_Version — 上报完整的栅格版本和构建信息

PostGIS 2.2.0dev 开发手册

378 / 686

**用法**

text PostGIS\_Raster\_Lib\_Version();

**描述**

上报完整的栅格版本和构建信息

**样例**

SELECT PostGIS\_Raster\_Lib\_Version();

postgis\_raster\_lib\_version

-----------------------------

2.0.0

**相关请参考**

PostGIS\_Lib\_Version

**9.2.6**

**ST\_GDALDrivers**

ST\_GDALDrivers — 返回你的GDAL库支持的栅格格式列表。这些格式是你使用函数ST\_AsGDALRaster所输出的类型

**用法**

setof record ST\_GDALDrivers(integer OUT idx, text OUT short\_name, text OUT long\_name, text OUT create\_options);

**描述**

返回你的GDAL库支持的栅格格式列表，包括格式简写、全称等等。函数ST\_AsGDALRaster使用的是简称。跟你编译GBDAL库所依赖的驱动类型，这些参数会有不同。create\_options参数作用是，根据每一个具体的驱动，一个xml格式（xml标签是：CreationOptionList/Option）的结果：包含名称、选项类型、描述

可用版本：2.0.0 -需要GDAL版本>= 1.6.0.

**样例: List of Drivers**

SELECT short\_name, long\_name

FROM st\_gdaldrivers()

ORDER BY short\_name;

short\_name

|

long\_name

----------------+--------------------------------------

AAIGrid

DTED

EHdr

FIT

GIF

GSAG

GSBG

GTiff

HF2

| Arc/Info ASCII Grid

| DTED Elevation Raster

| ESRI .hdr Labelled

| FIT Image

| Graphics Interchange Format (.gif)

| Golden Software ASCII Grid (.grd)

| Golden Software Binary Grid (.grd)

| GeoTIFF

| HF2/HFZ heightfield raster

PostGIS 2.2.0dev 开发手册

379 / 686

HFA

ILWIS

INGR

JPEG

KMLSUPEROVERLAY

NITF

PNG

R

SAGA

SRTMHGT

USGSDEM

VRT

XPM

|

|

|

|

|

|

|

|

|

|

|

|

|

Erdas Imagine Images (.img)

ILWIS Raster Map

Intergraph Raster

JPEG JFIF

Kml Super Overlay

National Imagery Transmission Format

Portable Network Graphics

R Object Data Store

SAGA GIS Binary Grid (.sdat)

SRTMHGT File Format

USGS Optional ASCII DEM (and CDED)

Virtual Raster

X11 PixMap Format

**样例: List of options for each driver**

-- Output the create options XML column of JPEG as a table

--

-- Note you can use these creator options in ST\_AsGDALRaster options argument

SELECT (xpath(’@name’, g.opt))[1]::text As oname,

(xpath(’@type’, g.opt))[1]::text As otype,

(xpath(’@description’, g.opt))[1]::text As descrip

FROM (SELECT unnest(xpath(’/CreationOptionList/Option’, create\_options::xml)) As opt

FROM

st\_gdaldrivers()

WHERE short\_name = ’JPEG’) As g;

oname

|

otype

|

descrip

-------------+---------+-----------------------------

PROGRESSIVE | boolean |

QUALITY

| int

| good=100, bad=0, default=75

WORLDFILE

| boolean |

-- raw xml output for creator options for GeoTiff --

SELECT create\_options

FROM st\_gdaldrivers()

WHERE short\_name = ’GTiff’;

<CreationOptionList>

<Option name="COMPRESS" type="string-select">

<Value>NONE</Value>

<Value>LZW</Value>

<Value>PACKBITS</Value>

<Value>JPEG</Value>

<Value>CCITTRLE</Value>

<Value>CCITTFAX3</Value>

<Value>CCITTFAX4</Value>

<Value>DEFLATE</Value>

</Option>

<Option name="PREDICTOR" type="int" description="Predictor Type"/>

<Option name="JPEG\_QUALITY" type="int" description="JPEG quality 1-100" default="75"/>

<Option name="ZLEVEL" type="int" description="DEFLATE compression level 1-9" default ←

="6"/>

<Option name="NBITS" type="int" description="BITS for sub-byte files (1-7), sub-uint16

(9-15), sub-uint32 (17-31)"/>

<Option name="INTERLEAVE" type="string-select" default="PIXEL">

<Value>BAND</Value>

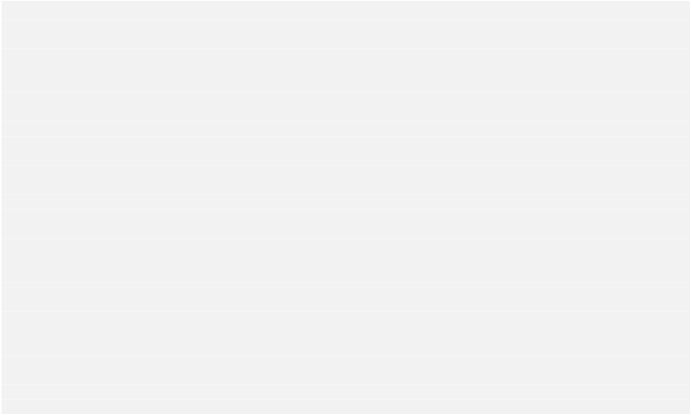
<Value>PIXEL</Value>

</Option>

<Option name="TILED" type="boolean" description="Switch to tiled format"/>

<Option name="TFW" type="boolean" description="Write out world file"/>

←



PostGIS 2.2.0dev 开发手册

380 / 686

<Option name="RPB" type="boolean" description="Write out .RPB (RPC) file"/>

<Option name="BLOCKXSIZE" type="int" description="Tile Width"/>

<Option name="BLOCKYSIZE" type="int" description="Tile/Strip Height"/>

<Option name="PHOTOMETRIC" type="string-select">

<Value>MINISBLACK</Value>

<Value>MINISWHITE</Value>

<Value>PALETTE</Value>

<Value>RGB</Value>

<Value>CMYK</Value>

<Value>YCBCR</Value>

<Value>CIELAB</Value>

<Value>ICCLAB</Value>

<Value>ITULAB</Value>

</Option>

<Option name="SPARSE\_OK" type="boolean" description="Can newly created files have

←

missing blocks?" default="FALSE"/>

<Option name="ALPHA" type="boolean" description="Mark first extrasample as being alpha ←

"/>

<Option name="PROFILE" type="string-select" default="GDALGeoTIFF">

<Value>GDALGeoTIFF</Value>

<Value>GeoTIFF</Value>

<Value>BASELINE</Value>

</Option>

<Option name="PIXELTYPE" type="string-select">

<Value>DEFAULT</Value>

<Value>SIGNEDBYTE</Value>

</Option>

<Option name="BIGTIFF" type="string-select" description="Force creation of BigTIFF file ←

">

<Value>YES</Value>

<Value>NO</Value>

<Value>IF\_NEEDED</Value>

<Value>IF\_SAFER</Value>

</Option>

<Option name="ENDIANNESS" type="string-select" default="NATIVE" description="Force

endianness of created file. For DEBUG purpose mostly">

<Value>NATIVE</Value>

<Value>INVERTED</Value>

<Value>LITTLE</Value>

<Value>BIG</Value>

</Option>

←

<Option name="COPY\_SRC\_OVERVIEWS" type="boolean" default="NO" description="Force copy

of overviews of source dataset (CreateCopy())"/>

</CreationOptionList>

←

-- Output the create options XML column for GTiff as a table

--

SELECT (xpath(’@name’, g.opt))[1]::text As oname,

(xpath(’@type’, g.opt))[1]::text As otype,

(xpath(’@description’, g.opt))[1]::text As descrip,

array\_to\_string(xpath(’Value/text()’, g.opt),’, ’) As vals

FROM (SELECT unnest(xpath(’/CreationOptionList/Option’, create\_options::xml)) As opt

FROM

st\_gdaldrivers()

WHERE short\_name = ’GTiff’) As g;

oname

|

otype

|

descrip

←

|

vals

--------------------+---------------+----------------------------------------------------------------

COMPRESS

| string-select |

←

| NONE, LZW,

←

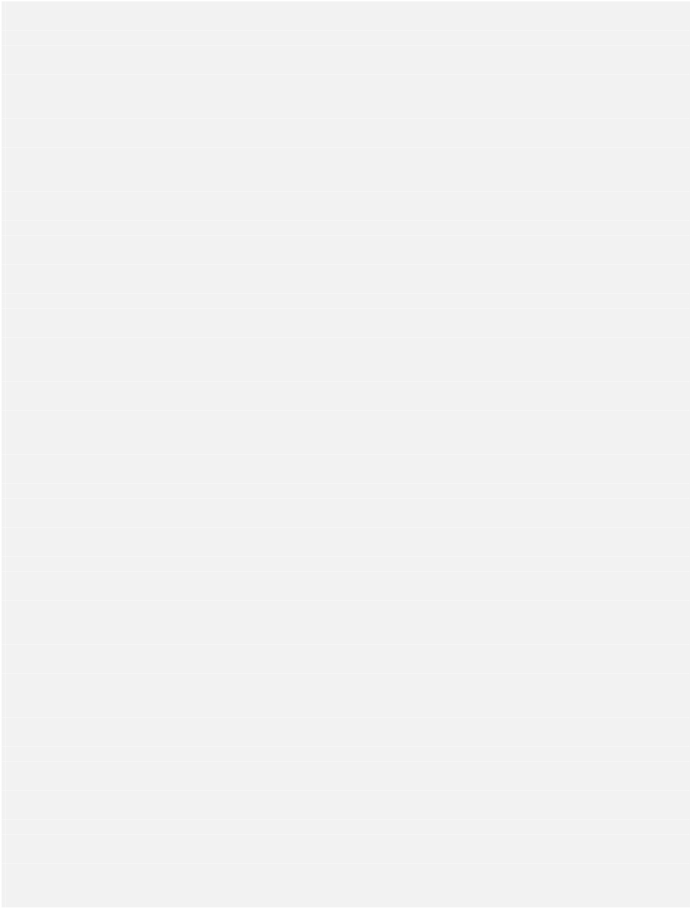
PACKBITS, JPEG, CCITTRLE, CCITTFAX3, CCITTFAX4, DEFLATE

PREDICTOR

| int

| Predictor Type

←



PostGIS 2.2.0dev 开发手册

381 / 686

|

JPEG\_QUALITY

| int

| JPEG quality 1-100

←

|

ZLEVEL

| int

| DEFLATE compression level 1-9

←

|

NBITS

| int

| BITS for sub-byte files (1-7), sub-uint16 (9-15), sub ←

-uint32 (17-31) |

INTERLEAVE

| string-select |

←

| BAND, PIXEL

TILED

| boolean

| Switch to tiled format

←

|

TFW

|

boolean

| Write out world file

←

|

RPB

|

boolean

| Write out .RPB (RPC) file

←

|

BLOCKXSIZE

|

int

| Tile Width

←

|

BLOCKYSIZE

|

int

| Tile/Strip Height

←

|

PHOTOMETRIC

|

string-select |

←

| MINISBLACK,

←

MINISWHITE, PALETTE, RGB, CMYK, YCBCR, CIELAB, ICCLAB, ITULAB

SPARSE\_OK

| boolean

| Can newly created files have missing blocks?

←

|

ALPHA

| boolean

| Mark first extrasample as being alpha

←

|

PROFILE

| string-select |

←

| GDALGeoTIFF,

←

GeoTIFF, BASELINE

PIXELTYPE

SIGNEDBYTE

| string-select |

←

| DEFAULT,

←

BIGTIFF

| string-select | Force creation of BigTIFF file

←

| YES, NO, IF\_NEEDED, IF\_SAFER

ENDIANNESS

mostly

| string-select | Force endianness of created file. For DEBUG purpose

| NATIVE, INVERTED, LITTLE, BIG

←

COPY\_SRC\_OVERVIEWS | boolean

()) |

(19 rows)

**相关请参考**

ST\_AsGDALRaster, ST\_SRID

| Force copy of overviews of source dataset (CreateCopy ←

**9.2.7**

**UpdateRasterSRID**

UpdateRasterSRID — 根据用户指定的表名和列名，修改栅格数据的SRID值

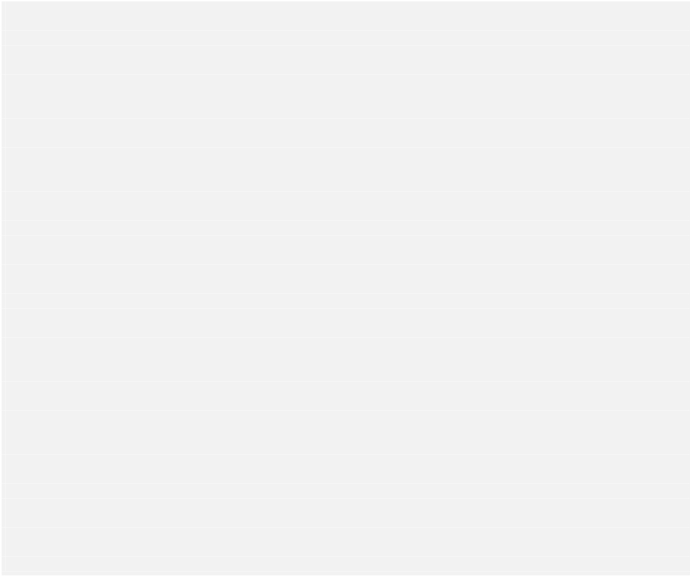
**用法**

raster UpdateRasterSRID(name schema\_name, name table\_name, name column\_name, integer new\_srid);

raster UpdateRasterSRID(name table\_name, name column\_name, integer new\_srid);

**描述**

根据用户指定的表名和列名，修改栅格数据的SRID值。在修改具体列的栅格SRID值前，该函数会删除该栅格列的所有约束。



PostGIS 2.2.0dev 开发手册

382 / 686

**注意**

该函数不会影响栅格的数据（波段的像素值），只会修改栅格的元数据信息（即SRID值）

可用版本：2.1.0

**9.3**

**9.3.1**

**栅格构建函数**

**ST\_AddBand**

ST\_AddBand —根据给定的类型和初始值以及波段位置，返回一个带有新波段的栅格，如果波段位置没有指定，则新波段添加到栅格末尾

**用法**

(1) raster ST\_AddBand(raster rast, addbandarg[] addbandargset);

(2) raster ST\_AddBand(raster rast, integer index, text pixeltype, double precision initialvalue=0, double precision nodataval=NULL);

(3) raster ST\_AddBand(raster rast, text pixeltype, double precision initialvalue=0, double precision nodataval=NULL);

(4) raster ST\_AddBand(raster torast, raster fromrast, integer fromband=1, integer torastindex=at\_end);

(5) raster ST\_AddBand(raster torast, raster[] fromrasts, integer fromband=1, integer torastindex=at\_end);

(6) raster ST\_AddBand(raster rast, integer index, text outdbﬁle, integer[] outdbindex, double precision nodataval=NULL);

(7) raster ST\_AddBand(raster rast, text outdbﬁle, integer[] outdbindex, integer index=at\_end, double precision nodataval=NULL);

**描述**

根据给定的类型和初始值以及波段位置，返回一个带有新波段的栅格，如果波段位置没有指定，则新波段添加到栅格末尾。如果没有指定fromband值，则默认取值为1。像元类型是函数ST\_BandPixelType返回的一段字符串。如果已有的位置索引已经指定，那么随后添加的波段位置索引将每次自增1。

如果指定了比像元类型最大还要大的初始值，那么初始值将被设定为所允许的像元类型的最大值。

对于那些使用数组类型参数addbandarg的函数(例如函数变体1)，一个指定的并且和栅格相关的位置索引值，在这个addbandarg表示的波段添加到栅格中时候，将会被设定。参考下面的多波段样例。

对于那些使用栅格数组类型的函数 (函数变体5),如果torast参数为NULL，那么参数数组中的每个栅格的fromband波段将被集中起来生成一个新波段。

对于那些使用outdbfile参数的函数 (函数变体6 和 7),这个参数值必须是栅格文件的绝对路径，并且这个栅格数据文件必须也能够让postgres的服务器进程能够访问到（译者注：就是有访问权限）

版本提升： 2.1.0 添加对addbandarg支持

版本提升： 2.1.0添加对库外波段支持，及形式7的支持

**样例: Single New Band**

-- Add another band of type 8 bit unsigned integer with pixels initialized to 200

UPDATE dummy\_rast

SET rast = ST\_AddBand(rast,’8BUI’::text,200)

WHERE rid = 1;



PostGIS 2.2.0dev 开发手册

383 / 686

-- Create an empty raster 100x100 units, with upper left

right at 0, add 2 bands (band 1

←

is 0/1 boolean bit switch, band2 allows values 0-15)

-- uses addbandargs

INSERT INTO dummy\_rast(rid,rast)

VALUES(10, ST\_AddBand(ST\_MakeEmptyRaster(100, 100, 0, 0, 1, -1, 0, 0, 0),

ARRAY[

ROW(1, ’1BB’::text, 0, NULL),

ROW(2, ’4BUI’::text, 0, NULL)

]::addbandarg[]

)

);

-- output meta data of raster bands to verify all is right --

SELECT

(bmd).\*

FROM (SELECT ST\_BandMetaData(rast,generate\_series(1,2)) As bmd

FROM dummy\_rast WHERE rid = 10) AS foo;

--result --

pixeltype | nodatavalue | isoutdb | path

-----------+----------------+-------------+---------+------

1BB

4BUI

| | f

| | f

|

|

-- output meta data of raster -

SELECT

(rmd).width, (rmd).height, (rmd).numbands

FROM (SELECT ST\_MetaData(rast) As rmd

FROM dummy\_rast WHERE rid = 10) AS foo;

-- result --

upperleftx | upperlefty | width | height | scalex | scaley | skewx | skewy | srid |

numbands

←

------------+------------+-------+--------+------------+------------+-------+-------+------+---------

0 |

0 |

100 |

100 |

1 |

-1 |

0 |

0 |

0 |

←

2

**样例: Multiple New Bands**

SELECT

\*

FROM ST\_BandMetadata(

ST\_AddBand(

ST\_MakeEmptyRaster(10, 10, 0, 0, 1, -1, 0, 0, 0),

ARRAY[

ROW(NULL, ’8BUI’, 255, 0),

ROW(NULL, ’16BUI’, 1, 2),

ROW(2, ’32BUI’, 100, 12),

ROW(2, ’32BF’, 3.14, -1)

]::addbandarg[]

),

ARRAY[]::integer[]

);

bandnum | pixeltype | nodatavalue | isoutdb | path

---------+-----------+-------------+---------+------

1 | 8BUI

2 | 32BF

3 | 32BUI

4 | 16BUI

| 0 | f

| -1 | f

| 12 | f

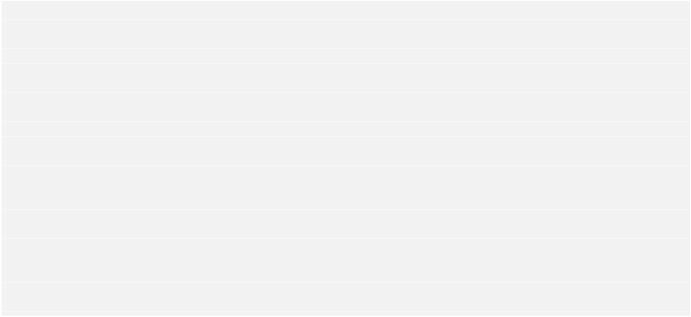
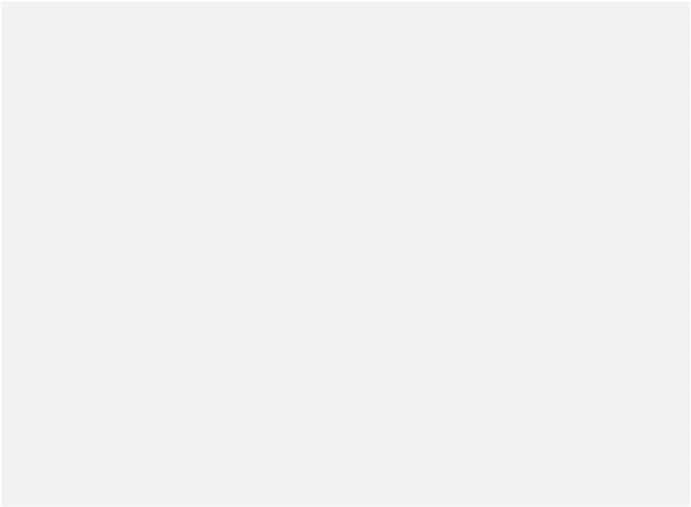
| 2 | f

|

|

|

|



PostGIS 2.2.0dev 开发手册

384 / 686

-- Aggregate the 1st band of a table of like rasters into a single raster

-- with as many bands as there are test\_types and as many rows (new rasters) as there are

←

mice

-- NOTE: The ORDER BY test\_type is only supported in PostgreSQL 9.0+

-- for 8.4 and below it usually works to order your data in a subselect (but not guaranteed ←

)

-- The resulting raster will have a band for each test\_type alphabetical by test\_type

-- For mouse lovers: No mice were harmed in this exercise

SELECT

mouse,

ST\_AddBand(NULL, array\_agg(rast ORDER BY test\_type), 1) As rast

FROM mice\_studies

GROUP BY mouse;

**样例: New Out-db band**

SELECT

\*

FROM ST\_BandMetadata(

ST\_AddBand(

ST\_MakeEmptyRaster(10, 10, 0, 0, 1, -1, 0, 0, 0),

’/home/raster/mytestraster.tif’::text, NULL::int[]

),

ARRAY[]::integer[]

);

bandnum | pixeltype | nodatavalue | isoutdb | path

---------+-----------+-------------+---------+------

1 | 8BUI

2 | 8BUI

3 | 8BUI

| | t

| | t

| | t

| /home/raster/mytestraster.tif

| /home/raster/mytestraster.tif

| /home/raster/mytestraster.tif

**相关请参考**

ST\_BandMetaData, ST\_BandPixelType, ST\_MakeEmptyRaster, ST\_MetaData, ST\_NumBands, ST\_Reclass

**9.3.2**

**ST\_AsRaster**

ST\_AsRaster — 把PostGIS 的几何类型对象转换成PostGIS的栅格对象

**用法**

raster ST\_AsRaster(geometry geom, raster ref, text pixeltype, double precision value=1, double precision nodataval=0, boolean

touched=false);

raster ST\_AsRaster(geometry geom, raster ref, text[] pixeltype=ARRAY[’8BUI’], double precision[] value=ARRAY[1], double

precision[] nodataval=ARRAY[0], boolean touched=false);

raster ST\_AsRaster(geometry geom, double precision scalex, double precision scaley, double precision gridx, double preci-

sion gridy, text pixeltype, double precision value=1, double precision nodataval=0, double precision skewx=0, double precision

skewy=0, boolean touched=false);

raster ST\_AsRaster(geometry geom, double precision scalex, double precision scaley, double precision gridx=NULL, dou-

ble precision gridy=NULL, text[] pixeltype=ARRAY[’8BUI’], double precision[] value=ARRAY[1], double precision[] no-

dataval=ARRAY[0], double precision skewx=0, double precision skewy=0, boolean touched=false);

raster ST\_AsRaster(geometry geom, double precision scalex, double precision scaley, text pixeltype, double precision value=1,

double precision nodataval=0, double precision upperleftx=NULL, double precision upperlefty=NULL, double precision skewx=0,

PostGIS 2.2.0dev 开发手册

385 / 686

double precision skewy=0, boolean touched=false);

raster ST\_AsRaster(geometry geom, double precision scalex, double precision scaley, text[] pixeltype, double precision[]

value=ARRAY[1], double precision[] nodataval=ARRAY[0], double precision upperleftx=NULL, double precision upperlefty=NULL,

double precision skewx=0, double precision skewy=0, boolean touched=false);

raster ST\_AsRaster(geometry geom, integer width, integer height, double precision gridx, double precision gridy, text pixel-

type, double precision value=1, double precision nodataval=0, double precision skewx=0, double precision skewy=0, boolean

touched=false);

raster ST\_AsRaster(geometry geom, integer width, integer height, double precision gridx=NULL, double precision gridy=NULL,

text[] pixeltype=ARRAY[’8BUI’], double precision[] value=ARRAY[1], double precision[] nodataval=ARRAY[0], double pre-

cision skewx=0, double precision skewy=0, boolean touched=false);

raster ST\_AsRaster(geometry geom, integer width, integer height, text pixeltype, double precision value=1, double precision

nodataval=0, double precision upperleftx=NULL, double precision upperlefty=NULL, double precision skewx=0, double preci-

sion skewy=0, boolean touched=false);

raster ST\_AsRaster(geometry geom, integer width, integer height, text[] pixeltype, double precision[] value=ARRAY[1], dou-

ble precision[] nodataval=ARRAY[0], double precision upperleftx=NULL, double precision upperlefty=NULL, double precision

skewx=0, double precision skewy=0, boolean touched=false);

**描述**

把PostGIS 的几何类型对象转换成PostGIS的栅格对象。该函数的不同形式提供了三组方式用于设置返回栅格像元的对齐方式和像元大小

第一组函数变体包含前两个变体。它们根据提供参考栅格生成包含相同对齐方式的栅格 (scalex, scaley, gridx ，

gridy),像元类型和NODATA值。一般情况下你可以通过表之间的join方式，来传递在该表中的参考栅格。

第二组函数变体包含了四个变体。可以让你通过像元的大小(scalex & scaley and skewx & skewy)来设置栅格的维度。返回的栅格的高度和宽度会被调整以便适应几何对象的边界。在大多数情况下，你必须把整型的scalex & scaley参数转换成double双精度型，以便PostgreSQL可以选择正确的函数变体。

第三组函数变体包含了四个变体，可以通过提供栅格（width和height参数）来设置栅格的维度。返回栅格的像元参数（scalex & scaley and skewx & skewy）会被调整以便适应几何对象的边界。

最后两组函数变体（即第二和第三组）的前两个变体都可以让你指定对齐网格(gridx & gridy)的任意一个角点的对齐方式，而后两个函数变体可以设置左上角的参数值 (upperleftx & upperlefty)。

每一组函数变体允许生成单波段栅格或者多波段栅格。若想生成多波段栅格，你必须提供一个像元类型数组(pixeltype[])，一个初始化值的数组，和一个NODATA值（用于无效值填充）数组。如果没有提供像元类型，默认值是8BUI，value参数的值为1，而参数nodataval默认值为0。

输出栅格将会和源几何对象在同一个参考系中。唯一的区别是带有参考栅格的函数变体，返回的栅格将会带有一个和参考栅格有相同SRID值。

可选参数touched会默认设置为false，并且和GDAL ALL\_TOUCHED 栅格化参数值一致，这个参数决定了像元是否会和所接触的lines 或 polygons对象融合。

这个函数在与函数ST\_AsPNG和其他ST\_AsGDALRaster系列函数联合使用，从数据库中直接导出几何对象的jpeg或png文件时候特别有用

可用版本：2.0.0 -需要GDAL版本>= 1.6.0.

**注意**

当前还不支持复杂的几何类型，比如curves, TINS, 和 PolyhedralSurfaces，但一旦GDAL支持了，这个函数也将会支持



PostGIS 2.2.0dev 开发手册

386 / 686

**样例: Output geometries as PNG ﬁles**

black circle

-- this will output a black circle taking up 150 x 150 pixels --

SELECT ST\_AsPNG(ST\_AsRaster(ST\_Buffer(ST\_Point(1,5),10),150, 150, ’2BUI’));

example from buffer rendered with just PostGIS

-- the bands map to RGB bands - the value (118,154,118) - teal

--

SELECT ST\_AsPNG(

ST\_AsRaster(

ST\_Buffer(

ST\_GeomFromText(’LINESTRING(50 50,150 150,150 50)’), 10,’join=bevel’),

200,200,ARRAY[’8BUI’, ’8BUI’, ’8BUI’], ARRAY[118,154,118], ARRAY[0,0,0]));

**相关请参考**

ST\_BandPixelType, ST\_Buffer, ST\_GDALDrivers, ST\_AsGDALRaster, ST\_AsPNG, ST\_AsJPEG, ST\_SRID

**9.3.3**

**ST\_Band**

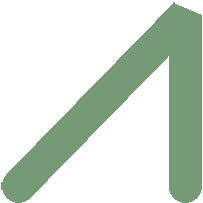
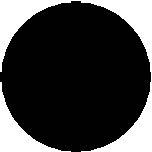
ST\_Band — 根据一个已经存在的栅格，返回一个或更多的波段当做新的栅格。用已有的栅格来创建新的栅格很有用

**用法**

raster ST\_Band(raster rast, integer[] nbands = ARRAY[1]);

raster ST\_Band(raster rast, text nbands, character delimiter=,);

raster ST\_Band(raster rast, integer nband);



PostGIS 2.2.0dev 开发手册

387 / 686

**描述**

根据一个已经存在的栅格，返回一个或更多的波段当做新的栅格。用已有的栅格来创建新的栅格或导出一个栅格中指定的波段很有用。如果没有选择波段，默认是波段1。在各种函数中用作一个辅助性函数，例如删除一个波段

可用版本：2.0.0

**样例**

-- Make 2 new rasters: 1 containing band 1 of dummy, second containing band 2 of dummy and

then reclassified as a 2BUI

SELECT ST\_NumBands(rast1) As numb1, ST\_BandPixelType(rast1) As pix1,

←

ST\_NumBands(rast2) As numb2,

ST\_BandPixelType(rast2) As pix2

FROM (

SELECT ST\_Band(rast) As rast1, ST\_Reclass(ST\_Band(rast,3), ’100-200):1, [200-254:2’, ’2 ←

BUI’) As rast2

FROM dummy\_rast

WHERE rid = 2) As foo;

numb1 | pix1 | numb2 | pix2

-------+------+-------+------

1 | 8BUI | 1 | 2BUI

-- Return bands 2 and 3. Use text to define bands

SELECT ST\_NumBands(ST\_Band(rast, ’2,3’)) As num\_bands

FROM dummy\_rast WHERE rid=2;

num\_bands

----------

2

-- Return bands 2 and 3. Use array to define bands

SELECT ST\_NumBands(ST\_Band(rast, ARRAY[2,3])) As num\_bands

FROM dummy\_rast

WHERE rid=2;

original (column rast)

dupe\_band

sing\_band

--Make a new raster with 2nd band of original and 1st band repeated twice,

and another with just the third band

SELECT rast, ST\_Band(rast, ARRAY[2,1,1]) As dupe\_band,



PostGIS 2.2.0dev 开发手册

388 / 686

ST\_Band(rast, 3) As sing\_band

FROM samples.than\_chunked

WHERE rid=35;

**相关请参考**

ST\_AddBand, ST\_NumBands, , ST\_Reclass

**9.3.4**

**ST\_MakeEmptyRaster**

ST\_MakeEmptyRaster — 根据给定的维度（宽度和高度），参考坐标或 X,Y 位置（通常在栅格左上角或左下角），像元大小和旋转信息(scalex, scaley, skewx & skewy) 和SRID值。如果传递了一个栅格对象，则返回一个新的栅格，新栅格和原栅格有一样的大小、对齐方式和SRID值。如果SRID值省略了，则SRID值设置为0（未知），返回一个空栅格（没有波段的栅格）

**用法**

raster ST\_MakeEmptyRaster(raster rast);

raster ST\_MakeEmptyRaster(integer width, integer height, ﬂoat8 upperleftx, ﬂoat8 upperlefty, ﬂoat8 scalex, ﬂoat8 scaley,

ﬂoat8 skewx, ﬂoat8 skewy, integer srid=unknown);

raster ST\_MakeEmptyRaster(integer width, integer height, ﬂoat8 upperleftx, ﬂoat8 upperlefty, ﬂoat8 pixelsize);

**描述**

根据给定的维度（宽度和高度），参考坐标或 X,Y 位置（通常在栅格左上角或左下角），像元大小和旋转信息(scalex, scaley, skewx & skewy) 和SRID值。如果传递了一个栅格对象，则返回一个新的栅格，新栅格和原栅格有一样的大小、对齐方式和SRID值，但没有波段。如果SRID值省略了，则SRID值设置为0（未知），返回一个空栅格（没有波段的栅格）。

最后一种形式使用单个参数来定义像元大小（pixelsize）。scalex用于定义这个参数，scaley定义该参数的负值。

skewx 和 skewy 设置为0

如果传递了一个栅格对象，则返回一个新的栅格，新栅格和原栅格有一样的大小、对齐方式和SRID值，但没有波段。

如果没有指定SRID值，默认设置为0，创建一个空栅格后，你可能需要添加波段并编辑它。

参考函数ST\_AddBand怎样定义波段和函数ST\_SetValue怎样初始化像元值

**样例**

INSERT INTO dummy\_rast(rid,rast)

VALUES(3, ST\_MakeEmptyRaster( 100, 100, 0.0005, 0.0005, 1, 1, 0, 0, 4326) );

--use an existing raster as template for new raster

INSERT INTO dummy\_rast(rid,rast)

SELECT 4, ST\_MakeEmptyRaster(rast)

FROM dummy\_rast WHERE rid = 3;

-- output meta data of rasters we just added

SELECT rid, (md).\*

FROM (SELECT rid, ST\_MetaData(rast) As md

FROM dummy\_rast

WHERE rid IN(3,4)) As foo;

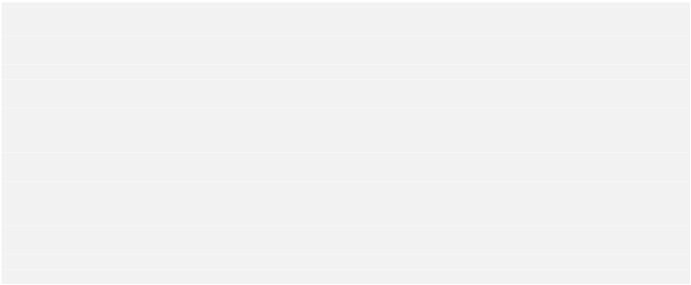
-- output --

rid | upperleftx | upperlefty | width | height | scalex | scaley | skewx | skewy | srid |

numbands

←

-----+------------+------------+-------+--------+------------+------------+-------+-------+------+---



PostGIS 2.2.0dev 开发手册

389 / 686

3 |

0.0005 |

0.0005 |

100 |

100 |

1 |

1 |

0

|

0 | ←

4326 |

0

4 |

0.0005 |

0.0005 |

100 |

100 |

1 |

1 |

0

|

0 | ←

4326 |

0

**相关请参考**

ST\_AddBand, ST\_MetaData, ST\_ScaleX, ST\_ScaleY, ST\_SetValue, ST\_SkewX, , ST\_SkewY

**9.3.5**

**ST\_Tile**

ST\_Tile —根据输入的栅格和输出的栅格维度，返回一个栅格集合

**用法**

setof raster ST\_Tile(raster rast, int[] nband, integer width, integer height, boolean padwithnodata=FALSE, double precision no-

dataval=NULL);

setof raster ST\_Tile(raster rast, integer nband, integer width, integer height, boolean padwithnodata=FALSE, double precision

nodataval=NULL);

setof raster ST\_Tile(raster rast, integer width, integer height, boolean padwithnodata=FALSE, double precision nodataval=NULL);

**描述**

根据输入的栅格和输出的栅格维度，返回一个栅格集合。

如果参数padwithnodata = FALSE, 栅格的右侧和左侧的边界瓦片可能和剩余瓦片的维度不同。如果参数padwithnodata = TRUE，那么所有的瓦片维度将会和填充的边界的瓦片维度相同，NODATA值也相同。如果栅格波段没有指定NODATA值，可以通过nodataval参数指定该值。

**注意**

如果输入栅格的指定波段在数据库之外，输出的栅格也将在数据库之外。（译者注：即文件形式）

可用版本：2.1.0

**样例**

WITH foo

AS (

SELECT

1,

SELECT

2,

SELECT

3,

SELECT

4,

SELECT

5,

SELECT

6,

ST\_AddBand(ST\_AddBand(ST\_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0), 1, ’8BUI’,

0), 2, ’8BUI’, 10, 0) AS rast UNION ALL

ST\_AddBand(ST\_AddBand(ST\_MakeEmptyRaster(3, 3, 3, 0, 1, -1, 0, 0, 0), 1, ’8BUI’,

0), 2, ’8BUI’, 20, 0) AS rast UNION ALL

ST\_AddBand(ST\_AddBand(ST\_MakeEmptyRaster(3, 3, 6, 0, 1, -1, 0, 0, 0), 1, ’8BUI’,

0), 2, ’8BUI’, 30, 0) AS rast UNION ALL

ST\_AddBand(ST\_AddBand(ST\_MakeEmptyRaster(3, 3, 0, -3, 1, -1, 0, 0, 0), 1, ’8BUI’,

0), 2, ’8BUI’, 40, 0) AS rast UNION ALL

ST\_AddBand(ST\_AddBand(ST\_MakeEmptyRaster(3, 3, 3, -3, 1, -1, 0, 0, 0), 1, ’8BUI’,

0), 2, ’8BUI’, 50, 0) AS rast UNION ALL

ST\_AddBand(ST\_AddBand(ST\_MakeEmptyRaster(3, 3, 6, -3, 1, -1, 0, 0, 0), 1, ’8BUI’,

0), 2, ’8BUI’, 60, 0) AS rast UNION ALL

←

←

←

←

←

←



PostGIS 2.2.0dev 开发手册

390 / 686

SELECT ST\_AddBand(ST\_AddBand(ST\_MakeEmptyRaster(3, 3, 0, -6, 1, -1, 0, 0, 0), 1, ’8BUI’,

7, 0), 2, ’8BUI’, 70, 0) AS rast UNION ALL

SELECT ST\_AddBand(ST\_AddBand(ST\_MakeEmptyRaster(3, 3, 3, -6, 1, -1, 0, 0, 0), 1, ’8BUI’,

8, 0), 2, ’8BUI’, 80, 0) AS rast UNION ALL

SELECT ST\_AddBand(ST\_AddBand(ST\_MakeEmptyRaster(3, 3, 6, -6, 1, -1, 0, 0, 0), 1, ’8BUI’,

9, 0), 2, ’8BUI’, 90, 0) AS rast

), bar AS (

SELECT ST\_Union(rast) AS rast FROM foo

), baz AS (

SELECT ST\_Tile(rast, 3, 3, TRUE) AS rast FROM bar

)

SELECT

ST\_DumpValues(rast)

FROM baz;

st\_dumpvalues

------------------------------------------

(1,"{{1,1,1},{1,1,1},{1,1,1}}")

(2,"{{10,10,10},{10,10,10},{10,10,10}}")

(1,"{{2,2,2},{2,2,2},{2,2,2}}")

(2,"{{20,20,20},{20,20,20},{20,20,20}}")

(1,"{{3,3,3},{3,3,3},{3,3,3}}")

(2,"{{30,30,30},{30,30,30},{30,30,30}}")

(1,"{{4,4,4},{4,4,4},{4,4,4}}")

(2,"{{40,40,40},{40,40,40},{40,40,40}}")

(1,"{{5,5,5},{5,5,5},{5,5,5}}")

(2,"{{50,50,50},{50,50,50},{50,50,50}}")

(1,"{{6,6,6},{6,6,6},{6,6,6}}")

(2,"{{60,60,60},{60,60,60},{60,60,60}}")

(1,"{{7,7,7},{7,7,7},{7,7,7}}")

(2,"{{70,70,70},{70,70,70},{70,70,70}}")

(1,"{{8,8,8},{8,8,8},{8,8,8}}")

(2,"{{80,80,80},{80,80,80},{80,80,80}}")

(1,"{{9,9,9},{9,9,9},{9,9,9}}")

(2,"{{90,90,90},{90,90,90},{90,90,90}}")

(18 rows)

←

←

←

WITH foo

AS (

SELECT

1,

SELECT

2,

SELECT

3,

SELECT

4,

SELECT

5,

SELECT

6,

ST\_AddBand(ST\_AddBand(ST\_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0), 1, ’8BUI’,

0), 2, ’8BUI’, 10, 0) AS rast UNION ALL

ST\_AddBand(ST\_AddBand(ST\_MakeEmptyRaster(3, 3, 3, 0, 1, -1, 0, 0, 0), 1, ’8BUI’,

0), 2, ’8BUI’, 20, 0) AS rast UNION ALL

ST\_AddBand(ST\_AddBand(ST\_MakeEmptyRaster(3, 3, 6, 0, 1, -1, 0, 0, 0), 1, ’8BUI’,

0), 2, ’8BUI’, 30, 0) AS rast UNION ALL

ST\_AddBand(ST\_AddBand(ST\_MakeEmptyRaster(3, 3, 0, -3, 1, -1, 0, 0, 0), 1, ’8BUI’,

0), 2, ’8BUI’, 40, 0) AS rast UNION ALL

ST\_AddBand(ST\_AddBand(ST\_MakeEmptyRaster(3, 3, 3, -3, 1, -1, 0, 0, 0), 1, ’8BUI’,

0), 2, ’8BUI’, 50, 0) AS rast UNION ALL

ST\_AddBand(ST\_AddBand(ST\_MakeEmptyRaster(3, 3, 6, -3, 1, -1, 0, 0, 0), 1, ’8BUI’,

0), 2, ’8BUI’, 60, 0) AS rast UNION ALL

←

←

←

←

←

←

SELECT ST\_AddBand(ST\_AddBand(ST\_MakeEmptyRaster(3, 3, 0, -6, 1, -1, 0, 0, 0), 1, ’8BUI’,

7, 0), 2, ’8BUI’, 70, 0) AS rast UNION ALL

SELECT ST\_AddBand(ST\_AddBand(ST\_MakeEmptyRaster(3, 3, 3, -6, 1, -1, 0, 0, 0), 1, ’8BUI’,

8, 0), 2, ’8BUI’, 80, 0) AS rast UNION ALL

SELECT ST\_AddBand(ST\_AddBand(ST\_MakeEmptyRaster(3, 3, 6, -6, 1, -1, 0, 0, 0), 1, ’8BUI’,

9, 0), 2, ’8BUI’, 90, 0) AS rast

), bar AS (

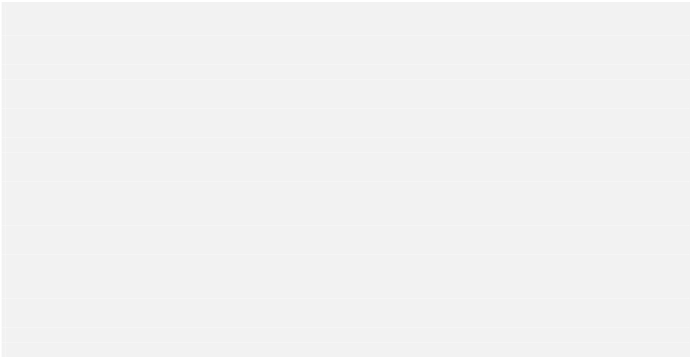
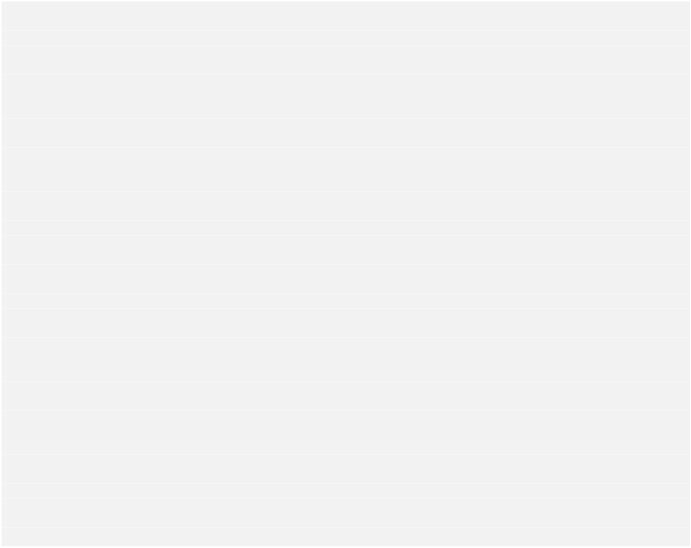
SELECT ST\_Union(rast) AS rast FROM foo

), baz AS (

←

←

←



PostGIS 2.2.0dev 开发手册

391 / 686

SELECT ST\_Tile(rast, 3, 3, 2) AS rast FROM bar

)

SELECT

ST\_DumpValues(rast)

FROM baz;

st\_dumpvalues

------------------------------------------

(1,"{{10,10,10},{10,10,10},{10,10,10}}")

(1,"{{20,20,20},{20,20,20},{20,20,20}}")

(1,"{{30,30,30},{30,30,30},{30,30,30}}")

(1,"{{40,40,40},{40,40,40},{40,40,40}}")

(1,"{{50,50,50},{50,50,50},{50,50,50}}")

(1,"{{60,60,60},{60,60,60},{60,60,60}}")

(1,"{{70,70,70},{70,70,70},{70,70,70}}")

(1,"{{80,80,80},{80,80,80},{80,80,80}}")

(1,"{{90,90,90},{90,90,90},{90,90,90}}")

(9 rows)

**相关请参考**

ST\_Union

**9.3.6**

**ST\_FromGDALRaster**

ST\_FromGDALRaster — 根据一个可以GDAL 支持的栅格数据，创建一个栅格

**用法**

raster ST\_FromGDALRaster(bytea gdaldata, integer srid=NULL);

**描述**

根据一个可以GDAL 支持的栅格数据，创建一个栅格。参数gdaldata 是bytea类型（译者注：bytea是PostgreSQL大型数据存储类型，相当于oracle的blob这样的类型），需要包含GDAL栅格数据的内容。

如果SRID值为NULL，该函数会从GDAL栅格中自行指定SRID值。如果SRID已经在参数中了，那么会覆盖GDAL栅格中的任何SRID值

可用版本：2.1.0

**样例**

WITH foo AS (

SELECT ST\_AsPNG(ST\_AddBand(ST\_AddBand(ST\_AddBand(ST\_MakeEmptyRaster(2, 2, 0, 0, 0.1,

-0.1, 0, 0, 4326), 1, ’8BUI’, 1, 0), 2, ’8BUI’, 2, 0), 3, ’8BUI’, 3, 0)) AS png

),

bar AS (

SELECT 1 AS rid, ST\_FromGDALRaster(png) AS rast FROM foo

UNION ALL

SELECT 2 AS rid, ST\_FromGDALRaster(png, 3310) AS rast FROM foo

)

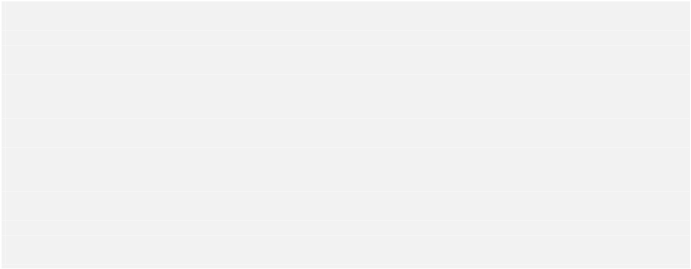
SELECT

rid,

ST\_Metadata(rast) AS metadata,

ST\_SummaryStats(rast, 1) AS stats1,

←



PostGIS 2.2.0dev 开发手册

392 / 686

ST\_SummaryStats(rast, 2) AS stats2,

ST\_SummaryStats(rast, 3) AS stats3

FROM bar

ORDER BY rid;

rid |

metadata

|

stats1

|

stats2

|

stats3

-----+---------------------------+---------------+---------------+----------------

1 | (0,0,2,2,1,-1,0,0,0,3) | (4,4,1,0,1,1) | (4,8,2,0,2,2) | (4,12,3,0,3,3)

2 | (0,0,2,2,1,-1,0,0,3310,3) | (4,4,1,0,1,1) | (4,8,2,0,2,2) | (4,12,3,0,3,3)

(2 rows)

**相关请参考**

ST\_AsGDALRaster

**9.4**

**9.4.1**

**栅格存取函数**

**ST\_GeoReference**

ST\_GeoReference — 返回栅格的地理参考元数据信息，有GDAL和ESRI格式，默认是GDAL格式

**用法**

text ST\_GeoReference(raster rast, text format=GDAL);

**描述**

在一个world file（关于该数据格式参考：https://en.wikipedia.org/wiki/World\_file）中返回栅格的地理参考元数据信息，有GDAL和ESRI格式，默认是GDAL格式。

GDAL和ESRI格式的区别如下：

GDAL:

scalex

skewy

skewx

scaley

upperleftx

upperlefty

ESRI:

scalex

skewy

skewx

scaley

upperleftx + scalex\*0.5

upperlefty + scaley\*0.5

PostGIS 2.2.0dev 开发手册

393 / 686

**样例**

SELECT ST\_GeoReference(rast, ’ESRI’) As esri\_ref, ST\_GeoReference(rast, ’GDAL’) As gdal\_ref

FROM dummy\_rast WHERE rid=1;

esri\_ref

|

gdal\_ref

--------------+--------------

2.0000000000 | 2.0000000000

0.0000000000 : 0.0000000000

0.0000000000 : 0.0000000000

3.0000000000 : 3.0000000000

1.5000000000 : 0.5000000000

2.0000000000 : 0.5000000000

**相关请参考**

ST\_SetGeoReference, ST\_ScaleX, ST\_ScaleY

**9.4.2**

**ST\_Height**

ST\_Height — 返回一个栅格的以像素数为单位的高度

**用法**

integer ST\_Height(raster rast);

**描述**

返回一个栅格的以像素数为单位的高度

**样例**

SELECT rid, ST\_Height(rast) As rastheight

FROM dummy\_rast;

rid | rastheight

-----+------------

1 | 20

2 | 5

**相关请参考**

ST\_Width

**9.4.3**

**ST\_IsEmpty**

ST\_IsEmpty — 如果栅格是空的(width = 0 and height = 0)，返回true，否则返回false

**用法**

boolean ST\_IsEmpty(raster rast);

PostGIS 2.2.0dev 开发手册

394 / 686

**描述**

如果栅格是空的(width = 0 and height = 0)，返回true，否则返回false

可用版本：2.0.0

**样例**

SELECT ST\_IsEmpty(ST\_MakeEmptyRaster(100, 100, 0, 0, 0, 0, 0, 0))

st\_isempty |

-----------+

f

|

SELECT ST\_IsEmpty(ST\_MakeEmptyRaster(0, 0, 0, 0, 0, 0, 0, 0))

st\_isempty |

-----------+

t

|

**相关请参考**

ST\_HasNoBand

**9.4.4**

**ST\_MetaData**

ST\_MetaData — 返回栅格对象的基本元数据信息，例如像元大小、旋转信息等等

**用法**

record ST\_MetaData(raster rast);

**描述**

返回栅格对象的基本元数据信息，例如像元大小、旋转信息等等。列的输出格式如下：

upperleftx | upperlefty | width | height | scalex | scaley | skewx | skewy | srid | numbands

**样例**

SELECT rid, (foo.md).\*

FROM (SELECT rid, ST\_MetaData(rast) As md

FROM dummy\_rast) As foo;

rid | upperleftx | upperlefty | width | height | scalex | scaley | skewx | skewy | srid |

←

numbands

----+------------+------------+-------+--------+--------+-----------+-------+-------+------+-------

1 |

0.5 |

0.5 |

10 |

20 |

2 |

3 |

0 |

0 |

0 |

←

0

2 | 3427927.75 |

5793244 |

5 |

5 |

0.05 |

-0.05 |

0 |

0 |

0 |

←

3

PostGIS 2.2.0dev 开发手册

395 / 686

**相关请参考**

ST\_BandMetaData, ST\_NumBands

**9.4.5**

**ST\_NumBands**

ST\_NumBands — 返回栅格对象的波段数

**用法**

integer ST\_NumBands(raster rast);

**描述**

返回栅格对象的波段数

**样例**

SELECT rid, ST\_NumBands(rast) As numbands

FROM dummy\_rast;

rid | numbands

----+----------

1 | 0

2 | 3

**相关请参考**

ST\_Value

**9.4.6**

**ST\_PixelHeight**

ST\_PixelHeight — 返回像元的高度，单位以空间参考系所规定的几何单位为准

**用法**

double precision ST\_PixelHeight(raster rast);

**描述**

返回像元的高度，单位以空间参考系所规定的几何单位为准。在没有倾斜的情况下，像素宽度只是几何坐标和栅格像素值的比值。

参考函数 ST\_PixelWidth来得到栅格关系的可视化结果

PostGIS 2.2.0dev 开发手册

396 / 686

**样例: Rasters with no skew**

SELECT ST\_Height(rast) As rastheight, ST\_PixelHeight(rast) As pixheight,

ST\_ScaleX(rast) As scalex, ST\_ScaleY(rast) As scaley, ST\_SkewX(rast) As skewx,

ST\_SkewY(rast) As skewy

FROM dummy\_rast;

rastheight | pixheight | scalex | scaley | skewx | skewy

------------+-----------+--------+--------+-------+----------

20 | 3 | 2 | 3 | 0 | 0

5 | 0.05 | 0.05 | -0.05 | 0 | 0

**样例: Rasters with skew different than 0**

SELECT ST\_Height(rast) As rastheight, ST\_PixelHeight(rast) As pixheight,

ST\_ScaleX(rast) As scalex, ST\_ScaleY(rast) As scaley, ST\_SkewX(rast) As skewx,

ST\_SkewY(rast) As skewy

FROM (SELECT ST\_SetSKew(rast,0.5,0.5) As rast

FROM dummy\_rast) As skewed;

rastheight |

pixheight

| scalex | scaley | skewx | skewy

-----------+-------------------+--------+--------+-------+----------

20 | 3.04138126514911 | 2 | 3 | 0.5 | 0.5

5 | 0.502493781056044 | 0.05 | -0.05 | 0.5 | 0.5

**相关请参考**

ST\_PixelWidth, ST\_ScaleX, ST\_ScaleY, ST\_SkewX, ST\_SkewY

**9.4.7**

**ST\_PixelWidth**

ST\_PixelWidth —返回像元的宽度，宽度单位以空间参考系规定的为准

**用法**

double precision ST\_PixelWidth(raster rast);

**描述**

返回像元的宽度，宽度单位以空间参考系规定的为准。在没有倾斜的情况下，像素宽度只是几何坐标和栅格像素值的比值。

下图展示了相关关系：

PostGIS 2.2.0dev 开发手册

397 / 686

Pixel Width: Pixel size in the i direction

Pixel Height: Pixel size in the j direction

**样例: Rasters with no skew**

SELECT ST\_Width(rast) As rastwidth, ST\_PixelWidth(rast) As pixwidth,

ST\_ScaleX(rast) As scalex, ST\_ScaleY(rast) As scaley, ST\_SkewX(rast) As skewx,

ST\_SkewY(rast) As skewy

FROM dummy\_rast;

rastwidth | pixwidth | scalex | scaley | skewx | skewy

-----------+----------+--------+--------+-------+----------

10 | 2 | 2 | 3 | 0 | 0

5 | 0.05 | 0.05 | -0.05 | 0 | 0

**样例: Rasters with skew different than 0**

SELECT ST\_Width(rast) As rastwidth, ST\_PixelWidth(rast) As pixwidth,

ST\_ScaleX(rast) As scalex, ST\_ScaleY(rast) As scaley, ST\_SkewX(rast) As skewx,

ST\_SkewY(rast) As skewy

FROM (SELECT ST\_SetSkew(rast,0.5,0.5) As rast

FROM dummy\_rast) As skewed;

rastwidth |

pixwidth

| scalex | scaley | skewx | skewy

-----------+-------------------+--------+--------+-------+----------

10 | 2.06155281280883 | 2 | 3 | 0.5 | 0.5

5 | 0.502493781056044 | 0.05 | -0.05 | 0.5 | 0.5

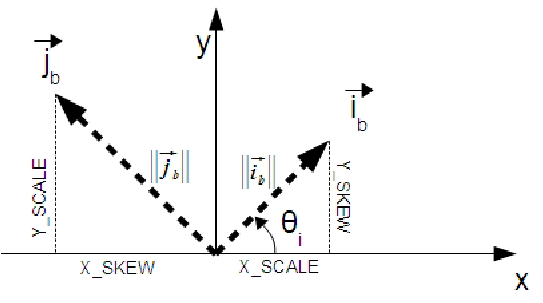
**相关请参考**

ST\_PixelHeight, ST\_ScaleX, ST\_ScaleY, ST\_SkewX, ST\_SkewY

**9.4.8**

**ST\_ScaleX**

ST\_ScaleX —像素块 x 方向上的像素数应输入整数值，即宽度，坐标单位以空间参考系为准



PostGIS 2.2.0dev 开发手册

398 / 686

**用法**

ﬂoat8 ST\_ScaleX(raster rast);

**描述**

像素块 x 方向上的像素数应输入整数值，坐标单位以空间参考系为准。参考World File获取更多信息

版本变更：2.0.0. 在WKTRaster 版本中，该函数叫ST\_PixelSizeX.

**样例**

SELECT rid, ST\_ScaleX(rast) As rastpixwidth

FROM dummy\_rast;

rid | rastpixwidth

-----+--------------

1 | 2

2 | 0.05

**相关请参考**

ST\_Width

**9.4.9**

**ST\_ScaleY**

ST\_ScaleY —像素块Y方向上的像素数应输入整数值，即高度，坐标单位以空间参考系为准

**用法**

ﬂoat8 ST\_ScaleY(raster rast);

**描述**

像素块Y方向上的像素数应输入整数值，即高度，坐标单位以空间参考系为准，参考World File获取更多信息

版本变更：2.0.0.在WKTRaster版本中，该函数被称为ST\_PixelSizeY.

**样例**

SELECT rid, ST\_ScaleY(rast) As rastpixheight

FROM dummy\_rast;

rid | rastpixheight

-----+---------------

1 | 3

2 | -0.05

**相关请参考**

ST\_Height

PostGIS 2.2.0dev 开发手册

399 / 686

**9.4.10**

**ST\_RasterToWorldCoord**

ST\_RasterToWorldCoord — 根据给定点的行和列的位置值，返回栅格的左上角点的几何值X和Y（即经度和维度），行和列从1开始。

**用法**

record ST\_RasterToWorldCoord(raster rast, integer xcolumn, integer yrow);

**描述**

根据给定点的行和列的位置值，返回栅格的左上角点的几何值X和Y（即经度和维度），行和列从1开始。返回的X和Y的值是根据参考栅格的几何单位定的。列和行的计数从1开始，但如果其中任意一个参数传递的参数是0或者一个负数或者大于参考栅格的维度值，那么返回的坐标会在栅格之外，因为这样的参数值会被假定超出了参考栅格的边界。

可用版本：2.1.0

**样例**

-- non-skewed raster

SELECT

rid,

(ST\_RasterToWorldCoord(rast,1, 1)).\*,

(ST\_RasterToWorldCoord(rast,2, 2)).\*

FROM dummy\_rast

rid | longitude

| latitude | longitude |

latitude

-----+------------+----------+-----------+------------

1 | 0.5 | 0.5 | 2.5 | 3.5

2 | 3427927.75 | 5793244 | 3427927.8 | 5793243.95

-- skewed raster

SELECT

rid,

(ST\_RasterToWorldCoord(rast, 1, 1)).\*,

(ST\_RasterToWorldCoord(rast, 2, 3)).\*

FROM (

SELECT

rid,

ST\_SetSkew(rast, 100.5, 0) As rast

FROM dummy\_rast

) As foo

rid | longitude

| latitude | longitude | latitude

-----+------------+----------+-----------+-----------

1 | 0.5 | 0.5 | 203.5 | 6.5

2 | 3427927.75 | 5793244 | 3428128.8 | 5793243.9

**相关请参考**

ST\_RasterToWorldCoordX, ST\_RasterToWorldCoordY, ST\_SetSkew

**9.4.11**

**ST\_RasterToWorldCoordX**

ST\_RasterToWorldCoordX —根据给定点的行和列的位置值，返回栅格的左上角点的X坐标（即经度），行和列从1开始。

PostGIS 2.2.0dev 开发手册

400 / 686

**用法**

ﬂoat8 ST\_RasterToWorldCoordX(raster rast, integer xcolumn);

ﬂoat8 ST\_RasterToWorldCoordX(raster rast, integer xcolumn, integer yrow);

**描述**

根据给定点的行和列的位置值，返回栅格的左上角点的X坐标（即经度），行和列从1开始。列和行的计数从1开始，但如果其中任意一个参数传递的参数是0或者一个负数或者大于参考栅格的维度值，那么返回的坐标会在栅格之外，因为这样的参数值会被假定超出了参考栅格的边界。

**注意**

对于非倾斜栅格，提供X列就足够了，对于倾斜栅格，其地理参考坐标是函数ST\_ScaleX 和 ST\_SkewX返回值，需要提供行和列的参数值。对于倾斜的栅格，如果值提供参数X值，那么将会抛出一个错误。

版本变更：2.1.0，在之前的版本中，该函数叫做ST\_Raster2WorldCoordX

**样例**

-- non-skewed raster providing column is sufficient

SELECT rid, ST\_RasterToWorldCoordX(rast,1) As x1coord,

ST\_RasterToWorldCoordX(rast,2) As x2coord,

ST\_ScaleX(rast) As pixelx

FROM dummy\_rast;

rid |

x1coord

|

x2coord

| pixelx

-----+------------+-----------+--------

1 | 0.5 | 2.5 | 2

2 | 3427927.75 | 3427927.8 | 0.05

-- for fun lets skew it

SELECT rid, ST\_RasterToWorldCoordX(rast, 1, 1) As x1coord,

ST\_RasterToWorldCoordX(rast, 2, 3) As x2coord,

ST\_ScaleX(rast) As pixelx

FROM (SELECT rid, ST\_SetSkew(rast, 100.5, 0) As rast FROM dummy\_rast) As foo;

rid |

x1coord

|

x2coord

| pixelx

-----+------------+-----------+--------

1 | 0.5 | 203.5 | 2

2 | 3427927.75 | 3428128.8 | 0.05

**相关请参考**

ST\_ScaleX, ST\_RasterToWorldCoordY, ST\_SetSkew, ST\_SkewX

**9.4.12**

**ST\_RasterToWorldCoordY**

ST\_RasterToWorldCoordY —根据给定点的行和列的位置值，返回栅格的左上角点的Y坐标（即维度），行和列从1开始。



PostGIS 2.2.0dev 开发手册

401 / 686

**用法**

ﬂoat8 ST\_RasterToWorldCoordY(raster rast, integer yrow);

ﬂoat8 ST\_RasterToWorldCoordY(raster rast, integer xcolumn, integer yrow);

**描述**

根据给定点的行和列的位置值，返回栅格的左上角点的Y坐标（即经度），行和列从1开始。列和行的计数从1开始，但如果其中任意一个参数传递的参数是0或者一个负数或者大于参考栅格的维度值，那么返回的坐标会在栅格之外，因为这样的参数值会被假定超出了参考栅格的边界。

**注意**

对于非倾斜栅格，提供Y参数就足够了，对于倾斜栅格，其地理参考坐标是函数ST\_ScaleY和 ST\_SkewY返回值，需要提供行和列的参数值。对于倾斜的栅格，如果值提供参数Y值，那么将会抛出一个错误。

版本变更：2.1.0 在之前的版本中，该函数叫做 ST\_Raster2WorldCoordY

**样例**

-- non-skewed raster providing row is sufficient

SELECT rid, ST\_RasterToWorldCoordY(rast,1) As y1coord,

ST\_RasterToWorldCoordY(rast,3) As y2coord,

ST\_ScaleY(rast) As pixely

FROM dummy\_rast;

rid | y1coord |

y2coord

| pixely

-----+---------+-----------+--------

1 | 0.5 | 6.5 | 3

2 | 5793244 | 5793243.9 | -0.05

-- for fun lets skew it

SELECT rid, ST\_RasterToWorldCoordY(rast,1,1) As y1coord,

ST\_RasterToWorldCoordY(rast,2,3) As y2coord,

ST\_ScaleY(rast) As pixely

FROM (SELECT rid, ST\_SetSkew(rast,0,100.5) As rast FROM dummy\_rast) As foo;

rid | y1coord |

y2coord

| pixely

-----+---------+-----------+--------

1 | 0.5 | 107 | 3

2 | 5793244 | 5793344.4 | -0.05

**相关请参考**

ST\_ScaleY, ST\_RasterToWorldCoordX, ST\_SetSkew, ST\_SkewY

**9.4.13**

**ST\_Rotation**

ST\_Rotation — 返回栅格的旋转角度，单位是弧度

**用法**

ﬂoat8 ST\_Rotation(raster rast);



PostGIS 2.2.0dev 开发手册

402 / 686

**描述**

返回栅格的旋转角度，单位是弧度。如果栅格旋转角度不唯一，那么返回NAN。参考World File（https://en.wikipedia.org/wiki/World\_file）获取更多信息

**样例**

SELECT rid, ST\_Rotation(ST\_SetScale(ST\_SetSkew(rast, sqrt(2)), sqrt(2))) as rot FROM

dummy\_rast;

←

rid |

rot

-----+-------------------

1 | 0.785398163397448

2 | 0.785398163397448

**相关请参考**

ST\_SetRotation, ST\_SetScale, ST\_SetSkew

**9.4.14**

**ST\_SkewX**

ST\_SkewX — 返回栅格的skew X值

**用法**

ﬂoat8 ST\_SkewX(raster rast);

**描述**

返回栅格的skew X值。参考World File（https://en.wikipedia.org/wiki/World\_file）获取更多信息.

**样例**

SELECT rid, ST\_SkewX(rast) As skewx, ST\_SkewY(rast) As skewy,

ST\_GeoReference(rast) as georef

FROM dummy\_rast;

rid | skewx | skewy |

georef

-----+-------+-------+--------------------

1 | 0 | 0 | 2.0000000000

: 0.0000000000

: 0.0000000000

: 3.0000000000

: 0.5000000000

: 0.5000000000

:

2 | 0 | 0 | 0.0500000000

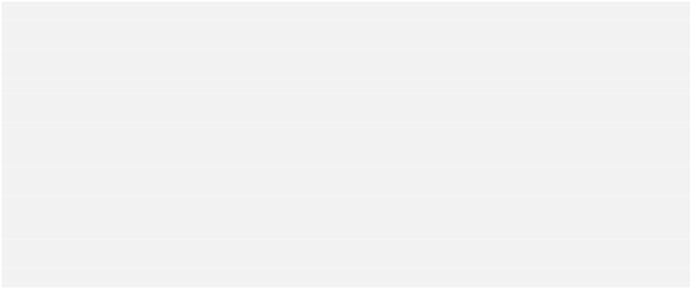
: 0.0000000000

: 0.0000000000

: -0.0500000000

: 3427927.7500000000

: 5793244.0000000000



PostGIS 2.2.0dev 开发手册

403 / 686

**相关请参考**

ST\_GeoReference, ST\_SkewY, ST\_SetSkew

**9.4.15**

**ST\_SkewY**

ST\_SkewY —返回栅格的skew Y值。

**用法**

ﬂoat8 ST\_SkewY(raster rast);

**描述**

返回栅格的skew Y值。参考World File（https://en.wikipedia.org/wiki/World\_file）获取更多信息.

**样例**

SELECT rid, ST\_SkewX(rast) As skewx, ST\_SkewY(rast) As skewy,

ST\_GeoReference(rast) as georef

FROM dummy\_rast;

rid | skewx | skewy |

georef

-----+-------+-------+--------------------

1 | 0 | 0 | 2.0000000000

: 0.0000000000

: 0.0000000000

: 3.0000000000

: 0.5000000000

: 0.5000000000

:

2 | 0 | 0 | 0.0500000000

: 0.0000000000

: 0.0000000000

: -0.0500000000

: 3427927.7500000000

: 5793244.0000000000

**相关请参考**

ST\_GeoReference, ST\_SkewX, ST\_SetSkew

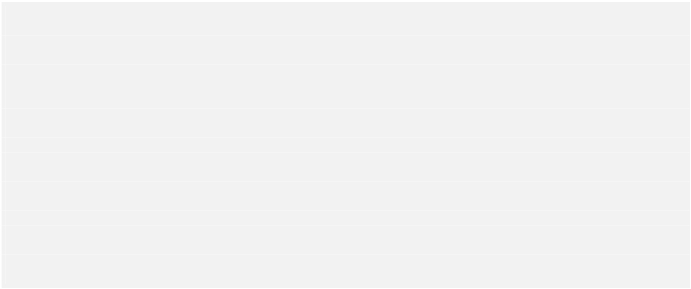
**9.4.16**

**ST\_SRID**

ST\_SRID — 返回栅格的SRID值

**用法**

integer ST\_SRID(raster rast);



PostGIS 2.2.0dev 开发手册

404 / 686

**描述**

返回栅格的SRID值

**注意**

从PostGIS 2.0+版本起，非地理参考系的栅格或geometry对象，SRID值为0，而不是之前版本的值-1

**样例**

SELECT ST\_SRID(rast) As srid

FROM dummy\_rast WHERE rid=1;

srid

----------------

0

**相关请参考**

Section 4.3.1, ST\_SRID

**9.4.17**

**ST\_Summary**

ST\_Summary — 返回一个栅格内容的文本描述

**用法**

text ST\_Summary(raster rast);

**描述**

返回一个栅格内容的文本描述

可用版本：2.1.0

**样例**

SELECT ST\_Summary(

ST\_AddBand(

ST\_AddBand(

ST\_AddBand(

ST\_MakeEmptyRaster(10, 10, 0, 0, 1, -1, 0, 0, 0)

, 1, ’8BUI’, 1, 0

)

, 2, ’32BF’, 0, -9999

)

, 3, ’16BSI’, 0, NULL

)

);

st\_summary

------------------------------------------------------------------



PostGIS 2.2.0dev 开发手册

405 / 686

Raster of 10x10 pixels has 3 bands and extent of BOX(0 -10,10 0)+

band 1 of pixtype 8BUI is in-db with NODATA value of 0 +

band 2 of pixtype 32BF is in-db with NODATA value of -9999 +

band 3 of pixtype 16BSI is in-db with no NODATA value

(1 row)

**相关请参考**

ST\_MetaData, ST\_BandMetaData, ST\_Summary ST\_Extent

**9.4.18**

**ST\_UpperLeftX**

ST\_UpperLeftX — 返回投影系中栅格的左上角X坐标

**用法**

ﬂoat8 ST\_UpperLeftX(raster rast);

**描述**

返回投影系中栅格的左上角X坐标

**样例**

SELECt rid, ST\_UpperLeftX(rast) As ulx

FROM dummy\_rast;

rid |

ulx

-----+------------

1 | 0.5

2 | 3427927.75

**相关请参考**

ST\_UpperLeftY, ST\_GeoReference, Box3D

**9.4.19**

**ST\_UpperLeftY**

ST\_UpperLeftY —返回投影系中栅格的左上角Y坐标

**用法**

ﬂoat8 ST\_UpperLeftY(raster rast);

**描述**

返回投影系中栅格的左上角Y坐标

PostGIS 2.2.0dev 开发手册

406 / 686

**样例**

SELECT rid, ST\_UpperLeftY(rast) As uly

FROM dummy\_rast;

rid |

uly

-----+---------

1 | 0.5

2 | 5793244

**相关请参考**

ST\_UpperLeftX, ST\_GeoReference, Box3D

**9.4.20**

**ST\_Width**

ST\_Width — 返回栅格的宽度，宽度以像素个数计数

**用法**

integer ST\_Width(raster rast);

**描述**

返回栅格的宽度，宽度以像素个数计数

**样例**

SELECT ST\_Width(rast) As rastwidth

FROM dummy\_rast WHERE rid=1;

rastwidth

----------------

10

**相关请参考**

ST\_Height

**9.4.21**

**ST\_WorldToRasterCoord**

ST\_WorldToRasterCoord — 根据给定X和Y值（通常是经度和维度），返回左上角的位置信息作为一条记录的行和列，或者栅格所在空间参考系中的一个点几何对象。

**用法**

record ST\_WorldToRasterCoord(raster rast, geometry pt);

record ST\_WorldToRasterCoord(raster rast, double precision longitude, double precision latitude);

PostGIS 2.2.0dev 开发手册

407 / 686

**描述**

根据给定X和Y值（通常是经度和维度），返回左上角的位置信息作为一条记录的行和列，或者栅格所在空间参考系中的一个点几何对象。无论几何位置（X，Y）或者Point几何对象是否在栅格上，该函数都可以使用。几何位置（X,Y）必须是栅格所在空间参考系所规定的格式

可用版本：2.1.0

**样例**

SELECT

rid,

(ST\_WorldToRasterCoord(rast,3427927.8,20.5)).\*,

(ST\_WorldToRasterCoord(rast,ST\_GeomFromText(’POINT(3427927.8 20.5)’,ST\_SRID(rast)))).\*

FROM dummy\_rast;

rid | columnx |

rowy

| columnx |

rowy

-----+---------+-----------+---------+-----------

1 | 1713964 | 7 | 1713964 | 7

2 | 2 | 115864471 | 2 | 115864471

**相关请参考**

ST\_WorldToRasterCoordX, ST\_WorldToRasterCoordY, ST\_RasterToWorldCoordX, ST\_RasterToWorldCoordY, ST\_SRID

**9.4.22**

**ST\_WorldToRasterCoordX**

ST\_WorldToRasterCoordX — 返回点几何对象在栅格中的列位置，或则栅格的空间参考系统中的X和Y空间坐标。

**用法**

integer ST\_WorldToRasterCoordX(raster rast, geometry pt);

integer ST\_WorldToRasterCoordX(raster rast, double precision xw);

integer ST\_WorldToRasterCoordX(raster rast, double precision xw, double precision yw);

**描述**

返回点几何对象在栅格中的列位置，或则栅格的空间参考系统中的X和Y空间坐标。如果栅格是倾斜的，那么参数pt或者参数xw/yw坐标是需要的。如果栅格是非倾斜的，那么参数xw就已经足够了。空间坐标的参考标准都是栅格所在的空间参考系。

版本变更：2.1.0 在之前的版本中，该函数叫做 ST\_World2RasterCoordX

**样例**

SELECT rid, ST\_WorldToRasterCoordX(rast,3427927.8) As xcoord,

ST\_WorldToRasterCoordX(rast,3427927.8,20.5) As xcoord\_xwyw,

ST\_WorldToRasterCoordX(rast,ST\_GeomFromText(’POINT(3427927.8 20.5)’,ST\_SRID(rast))) As

ptxcoord

FROM dummy\_rast;

←

rid | xcoord

|

xcoord\_xwyw

| ptxcoord

-----+---------+---------+----------

1 | 1713964 | 1713964 | 1713964

2 | 1 | 1 | 1

PostGIS 2.2.0dev 开发手册

408 / 686

**相关请参考**

ST\_RasterToWorldCoordX, ST\_RasterToWorldCoordY, ST\_SRID

**9.4.23**

**ST\_WorldToRasterCoordY**

ST\_WorldToRasterCoordY —返回点几何对象在栅格中的行位置，或则栅格的空间参考系统中的X和Y空间坐标。

**用法**

integer ST\_WorldToRasterCoordY(raster rast, geometry pt);

integer ST\_WorldToRasterCoordY(raster rast, double precision xw);

integer ST\_WorldToRasterCoordY(raster rast, double precision xw, double precision yw);

**描述**

返回点几何对象在栅格中的行位置，或则栅格的空间参考系统中的X和Y空间坐标。如果栅格是倾斜的，那么参数pt或者参数xw/yw坐标是需要的。如果栅格是非倾斜的，那么参数xw就已经足够了。空间坐标的参考标准都是栅格所在的空间参考系。

版本变更：2.1.0 在之前的版本中，该函数叫做 ST\_World2RasterCoordY

**样例**

SELECT rid, ST\_WorldToRasterCoordY(rast,20.5) As ycoord,

ST\_WorldToRasterCoordY(rast,3427927.8,20.5) As ycoord\_xwyw,

ST\_WorldToRasterCoordY(rast,ST\_GeomFromText(’POINT(3427927.8 20.5)’,ST\_SRID(rast))) As

ptycoord

FROM dummy\_rast;

←

rid |

ycoord

| ycoord\_xwyw | ptycoord

-----+-----------+-------------+-----------

1 | 7 | 7 | 7

2 | 115864471 | 115864471 | 115864471

**相关请参考**

ST\_RasterToWorldCoordX, ST\_RasterToWorldCoordY, ST\_SRID

**9.5**

**9.5.1**

**栅格波段存取函数**

**ST\_BandMetaData**

ST\_BandMetaData —返回指定的栅格波段的基本元数据信息，如果没有指定栅格波段，默认为1

**用法**

record ST\_BandMetaData(raster rast, integer bandnum=1);

PostGIS 2.2.0dev 开发手册

409 / 686

**描述**

返回指定的栅格波段的基本元数据信息，如果没有指定栅格波段，默认为1。返回的元数据个数如下：pixeltype | nodatavalue | isoutdb | path.

**注意**

如果栅格没有包含波段，那么将会抛出错误

**注意**

如果波段没有NODATA 值，那么NODATA值会被设置为NULL

**样例**

SELECT rid, (foo.md).\*

FROM (SELECT rid, ST\_BandMetaData(rast,1) As md

FROM dummy\_rast WHERE rid=2) As foo;

rid | pixeltype | nodatavalue | isoutdb | path

-----+-----------+----------------+-------------+---------+------

2 | 8BUI

| 0 | f

|

**相关请参考**

ST\_MetaData, ST\_BandPixelType

**9.5.2**

**ST\_BandNoDataValue**

ST\_BandNoDataValue —返回给定栅格波段的NODATA值，如果没有给出指定波段，则默认为取1。

**用法**

double precision ST\_BandNoDataValue(raster rast, integer bandnum=1);

**描述**

返回给定栅格波段的NODATA值，如果没有给出指定波段，则默认为取1。

**样例**

SELECT ST\_BandNoDataValue(rast,1) As bnval1,

ST\_BandNoDataValue(rast,2) As bnval2, ST\_BandNoDataValue(rast,3) As bnval3

FROM dummy\_rast

WHERE rid = 2;

bnval1 | bnval2 | bnval3

--------+--------+--------

0 | 0 | 0



PostGIS 2.2.0dev 开发手册

410 / 686

**相关请参考**

ST\_NumBands

**9.5.3**

**ST\_BandIsNoData**

ST\_BandIsNoData —如果波段只设置了NODATA值，则返回true

**用法**

boolean ST\_BandIsNoData(raster rast, integer band, boolean forceChecking=true);

boolean ST\_BandIsNoData(raster rast, boolean forceChecking=true);

**描述**

如果波段只设置了NODATA值，则返回true。如果没有指定波段位置，则默认为波段1。如果最后一个参数forceChecking值为true，那么整个波段将会一个像素一个像素进行校验；否则该函数简单返回该波段的的isnodata标识。如果没有指定该参数，那么默认参数值是FALSE

可用版本：2.0.0

**注意**

如果标识是“脏”的 (这里脏的意思是指，对同一个栅格同一个波段使用参数forceChecking值分别为true和false时候，得到的结果是不同的，那么就称是脏的。)，这时候你应该更新栅格，来保证返回值为true，更新方法是使用函数ST\_SetBandIsNodata()或函数 ST\_SetBandNodataValue()（最后一个参数值设置为true）。参考 ST\_SetBandIsNoData.

**样例**

-- Create dummy table with one raster column

create table dummy\_rast (rid integer, rast raster);

-- Add raster with two bands, one pixel/band. In the first band, nodatavalue = pixel value

= 3.

-- In the second band, nodatavalue = 13, pixel value = 4

insert into dummy\_rast values(1,

(

’01’ -- little endian (uint8 ndr)

||

’0000’ -- version (uint16 0)

||

’0200’ -- nBands (uint16 0)

||

’17263529ED684A3F’ -- scaleX (float64 0.000805965234044584)

||

’F9253529ED684ABF’ -- scaleY (float64 -0.00080596523404458)

||

’1C9F33CE69E352C0’ -- ipX (float64 -75.5533328537098)

||

’718F0E9A27A44840’ -- ipY (float64 49.2824585505576)

||

’ED50EB853EC32B3F’ -- skewX (float64 0.000211812383858707)

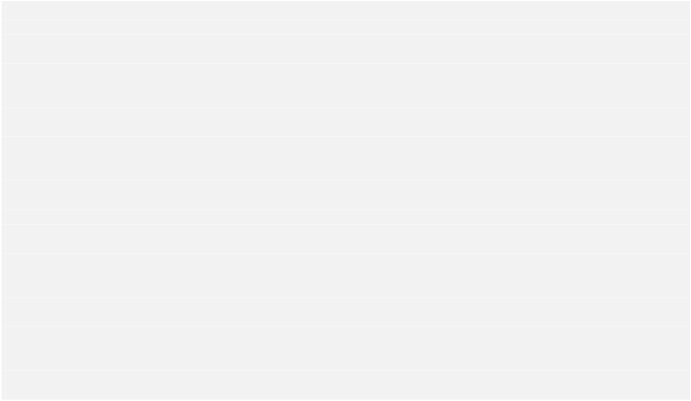
||

’7550EB853EC32B3F’ -- skewY (float64 0.000211812383858704)

||

’E6100000’ -- SRID (int32 4326)

←



PostGIS 2.2.0dev 开发手册

411 / 686

||

’0100’ -- width (uint16 1)

||

’0100’ -- height (uint16 1)

||

’6’ -- hasnodatavalue and isnodata value set to true.

||

’2’ -- first band type (4BUI)

||

’03’ -- novalue==3

||

’03’ -- pixel(0,0)==3 (same that nodata)

||

’0’ -- hasnodatavalue set to false

||

’5’ -- second band type (16BSI)

||

’0D00’ -- novalue==13

||

’0400’ -- pixel(0,0)==4

)::raster

);

select st\_bandisnodata(rast, 1) from dummy\_rast where rid = 1; -- Expected true

select st\_bandisnodata(rast, 2) from dummy\_rast where rid = 1; -- Expected false

**相关请参考**

ST\_BandNoDataValue, ST\_NumBands, ST\_SetBandNoDataValue, ST\_SetBandIsNoData

**9.5.4**

**ST\_BandPath**

ST\_BandPath — 返回存储在文件系统中的波段的文件路径。如果没有指定是哪个波段，默认是波段1。

**用法**

text ST\_BandPath(raster rast, integer bandnum=1);

**描述**

返回存储在文件系统中的波段的文件路径。如果没有指定是哪个波段，默认是波段1。如果波段是数据库中的波段，将抛出一个错误。

**样例**

**相关请参考**

**9.5.5**

**ST\_BandPixelType**

ST\_BandPixelType —返回给定波段的像素类型，如果没有指定是哪个波段，默认是波段1。

**用法**

text ST\_BandPixelType(raster rast, integer bandnum=1);



PostGIS 2.2.0dev 开发手册

412 / 686

**描述**

返回给定波段的像素类型，如果没有指定是哪个波段，默认是波段1。

总计有11种波段类型，支持的波段类型如下：

· 1BB - 1-bit boolean

· 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

· 32BF - 32-bit ﬂoat

· 64BF - 64-bit ﬂoat

**样例**

SELECT ST\_BandPixelType(rast,1) As btype1,

ST\_BandPixelType(rast,2) As btype2, ST\_BandPixelType(rast,3) As btype3

FROM dummy\_rast

WHERE rid = 2;

btype1 | btype2 | btype3

--------+--------+--------

8BUI

| 8BUI

| 8BUI

**相关请参考**

ST\_NumBands

**9.5.6**

**ST\_HasNoBand**

ST\_HasNoBand — 根据给定的波段，如果没有该波段，则返回true。如果没有指定波段，则默认指的是波段1。

**用法**

boolean ST\_HasNoBand(raster rast, integer bandnum=1);

**描述**

根据给定的波段，如果没有该波段，则返回true。如果没有指定波段，则默认指的是波段1。

可用版本：2.0.0

PostGIS 2.2.0dev 开发手册

413 / 686

**样例**

SELECT rid, ST\_HasNoBand(rast) As hb1, ST\_HasNoBand(rast,2) as hb2,

ST\_HasNoBand(rast,4) as hb4, ST\_NumBands(rast) As numbands

FROM dummy\_rast;

rid | hb1 | hb2 | hb4 | numbands

-----+-----+-----+-----+----------

1 | t

2 | f

| t

| f

| t

| t

| 0

| 3

**相关请参考**

ST\_NumBands

**9.6**

**9.6.1**

**Raster Pixel Accessors and Setters**

**ST\_PixelAsPolygon**

ST\_PixelAsPolygon — 返回包含指定行和列数的像素的多边形几何对象。

**用法**

geometry ST\_PixelAsPolygon(raster rast, integer columnx, integer rowy);

**描述**

返回包含指定行和列数的像素的多边形几何对象。

可用版本：2.0.0

**样例**

-- get raster pixel polygon

SELECT i,j, ST\_AsText(ST\_PixelAsPolygon(foo.rast, i,j)) As b1pgeom

FROM dummy\_rast As foo

CROSS JOIN generate\_series(1,2) As i

CROSS JOIN generate\_series(1,1) As j

WHERE rid=2;

i | j |

b1pgeom

---+---+-----------------------------------------------------------------------------

1 | 1 | POLYGON((3427927.75 5793244,3427927.8 5793244,3427927.8 5793243.95,...

2 | 1 | POLYGON((3427927.8 5793244,3427927.85 5793244,3427927.85 5793243.95, ..

**相关请参考**

ST\_DumpAsPolygons, ST\_PixelAsPolygons, ST\_PixelAsPoint, ST\_PixelAsPoints, ST\_PixelAsCentroid, ST\_PixelAsCentroids,

ST\_Intersection, ST\_AsText

PostGIS 2.2.0dev 开发手册

414 / 686

**9.6.2**

**ST\_PixelAsPolygons**

ST\_PixelAsPolygons —返回一个包含一个栅格波段的每个像元的多边形几何对象的记录集，该记录集包含列value，每个像元的X和Y栅格坐标。

**用法**

setof record ST\_PixelAsPolygons(raster rast, integer band=1, boolean exclude\_nodata\_value=TRUE);

**描述**

返回一个包含一个栅格波段的每个像元的多边形几何对象的记录集，该记录集包含列value，每个像元的X和Y栅格坐标。

**注意**

ST\_PixelAsPolygons 返回每个像元的多边形几何对象。这个函数和ST\_DumpAsPolygons是不同的，每个几何对象代表一个或更多像素值相同的几何对象。

**注意**

当exclude\_nodata\_value = TRUE时，只有像素值不是NODATA值的像素会返回多边形几何对象。

可用版本：2.0.0

版本提升： 2.1.0 添加参数exclude\_nodata\_value optional 支持

版本变更：2.1.1 修改了exclude\_nodata\_value值的实现

**样例**

-- get raster pixel polygon

SELECT (gv).x, (gv).y, (gv).val, ST\_AsText((gv).geom) geom

FROM (SELECT ST\_PixelAsPolygons(

) gv

) foo;

ST\_SetValue(ST\_SetValue(ST\_AddBand(ST\_MakeEmptyRaster(2, 2, 0, 0, 0.001,

-0.001, 0.001, 0.001, 4269),

’8BUI’::text, 1, 0),

2, 2, 10),

1, 1, NULL)

←

x | y | val |

geom

---+---+-----------------------------------------------------------------------------

1 | 1 | | POLYGON((0 0,0.001 0.001,0.002 0,0.001 -0.001,0 0))

1 | 2 | 1 | POLYGON((0.001 -0.001,0.002 0,0.003 -0.001,0.002 -0.002,0.001 -0.001))

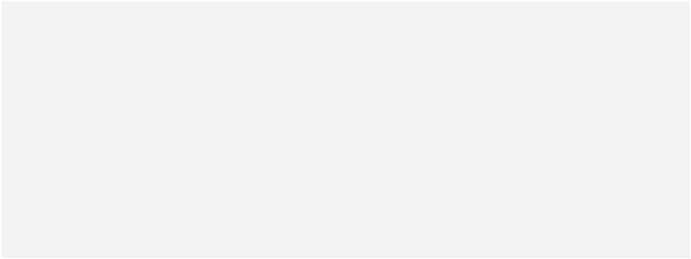
2 | 1 | 1 | POLYGON((0.001 0.001,0.002 0.002,0.003 0.001,0.002 0,0.001 0.001))

2 | 2 | 10 | POLYGON((0.002 0,0.003 0.001,0.004 0,0.003 -0.001,0.002 0))

**相关请参考**

ST\_DumpAsPolygons, ST\_PixelAsPolygon, ST\_PixelAsPoint, ST\_PixelAsPoints, ST\_PixelAsCentroid, ST\_PixelAsCentroids,

ST\_AsText



PostGIS 2.2.0dev 开发手册

415 / 686

**9.6.3**

**ST\_PixelAsPoint**

ST\_PixelAsPoint —返回像元左上角的point几何对象

**用法**

geometry ST\_PixelAsPoint(raster rast, integer columnx, integer rowy);

**描述**

返回像元左上角的point几何对象

可用版本：2.1.0

**样例**

SELECT ST\_AsText(ST\_PixelAsPoint(rast, 1, 1)) FROM dummy\_rast WHERE rid = 1;

st\_astext

----------------

POINT(0.5 0.5)

**相关请参考**

ST\_DumpAsPolygons, ST\_PixelAsPolygon, ST\_PixelAsPolygons, ST\_PixelAsPoints, ST\_PixelAsCentroid, ST\_PixelAsCentroids

**9.6.4**

**ST\_PixelAsPoints**

ST\_PixelAsPoints —返回一个包含一个栅格波段的每个像元的点几何对象的记录集，该记录集包含列value，每个像元的X和Y栅格坐标。点几何对象的坐标是像元的左上角。

**用法**

geometry ST\_PixelAsPoints(raster rast, integer band=1, boolean exclude\_nodata\_value=TRUE);

**描述**

返回一个包含一个栅格波段的每个像元的点几何对象的记录集，该记录集包含列value，每个像元的X和Y栅格坐标。点几何对象的坐标是像元的左上角。

**注意**

当exclude\_nodata\_value = TRUE,只有那些像素值不为NODATA值的像素返回点几何对象。

可用版本：2.1.0

版本变更：2.1.1 ，修改了exclude\_nodata\_value值对应的操作方式



PostGIS 2.2.0dev 开发手册

416 / 686

**样例**

SELECT x, y, val, ST\_AsText(geom) FROM (SELECT (ST\_PixelAsPoints(rast, 1)).\* FROM

dummy\_rast WHERE rid = 2) foo;

←

x | y | val |

st\_astext

---+---+-----+------------------------------

1 | 1 | 253 | POINT(3427927.75 5793244)

2 | 1 | 254 | POINT(3427927.8 5793244)

3 | 1 | 253 | POINT(3427927.85 5793244)

4 | 1 | 254 | POINT(3427927.9 5793244)

5 | 1 | 254 | POINT(3427927.95 5793244)

1 | 2 | 253 | POINT(3427927.75 5793243.95)

2 | 2 | 254 | POINT(3427927.8 5793243.95)

3 | 2 | 254 | POINT(3427927.85 5793243.95)

4 | 2 | 253 | POINT(3427927.9 5793243.95)

5 | 2 | 249 | POINT(3427927.95 5793243.95)

1 | 3 | 250 | POINT(3427927.75 5793243.9)

2 | 3 | 254 | POINT(3427927.8 5793243.9)

3 | 3 | 254 | POINT(3427927.85 5793243.9)

4 | 3 | 252 | POINT(3427927.9 5793243.9)

5 | 3 | 249 | POINT(3427927.95 5793243.9)

1 | 4 | 251 | POINT(3427927.75 5793243.85)

2 | 4 | 253 | POINT(3427927.8 5793243.85)

3 | 4 | 254 | POINT(3427927.85 5793243.85)

4 | 4 | 254 | POINT(3427927.9 5793243.85)

5 | 4 | 253 | POINT(3427927.95 5793243.85)

1 | 5 | 252 | POINT(3427927.75 5793243.8)

2 | 5 | 250 | POINT(3427927.8 5793243.8)

3 | 5 | 254 | POINT(3427927.85 5793243.8)

4 | 5 | 254 | POINT(3427927.9 5793243.8)

5 | 5 | 254 | POINT(3427927.95 5793243.8)

**相关请参考**

ST\_DumpAsPolygons, ST\_PixelAsPolygon, ST\_PixelAsPolygons, ST\_PixelAsPoint, ST\_PixelAsCentroid, ST\_PixelAsCentroids

**9.6.5**

**ST\_PixelAsCentroid**

ST\_PixelAsCentroid — 返回栅格某个像元的中心点（point geometry类型）

**用法**

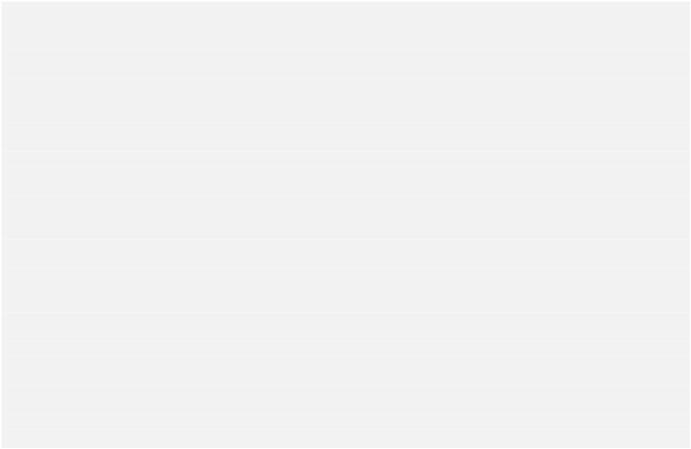
geometry ST\_PixelAsCentroid(raster rast, integer columnx, integer rowy);

**描述**

返回栅格某个像元的中心点（point geometry类型）

可用版本：2.1.0

**样例**



PostGIS 2.2.0dev 开发手册

417 / 686

SELECT ST\_AsText(ST\_PixelAsCentroid(rast, 1, 1)) FROM dummy\_rast WHERE rid = 1;

st\_astext

--------------

POINT(1.5 2)

**相关请参考**

ST\_DumpAsPolygons, ST\_PixelAsPolygon, ST\_PixelAsPolygons, ST\_PixelAsPoint, ST\_PixelAsPoints, ST\_PixelAsCentroids

**9.6.6**

**ST\_PixelAsCentroids**

ST\_PixelAsCentroids —返回一个包含一个栅格波段的中心（点几何对象）的点几何对象的记录集，该记录集包含列value，每个像元的X和Y栅格坐标。点几何对象表示的是像元的中心。

**用法**

geometry ST\_PixelAsCentroids(raster rast, integer band=1, boolean exclude\_nodata\_value=TRUE);

**描述**

返回一个包含一个栅格波段的中心（点几何对象）的点几何对象的记录集，该记录集包含列value，每个像元的X和Y栅格坐标。点几何对象表示的是像元的中心。

**注意**

当exclude\_nodata\_value = TRUE,只有那些像素值不为NODATA值的像素返回点几何对象。

可用版本：2.1.0

版本变更：2.1.1 ，修改了exclude\_nodata\_value值对应的操作方式

**样例**

SELECT x, y, val, ST\_AsText(geom) FROM (SELECT (ST\_PixelAsCentroids(rast, 1)).\* FROM

dummy\_rast WHERE rid = 2) foo;

←

x | y | val |

st\_astext

---+---+-----+--------------------------------

1 | 1 | 253 | POINT(3427927.775 5793243.975)

2 | 1 | 254 | POINT(3427927.825 5793243.975)

3 | 1 | 253 | POINT(3427927.875 5793243.975)

4 | 1 | 254 | POINT(3427927.925 5793243.975)

5 | 1 | 254 | POINT(3427927.975 5793243.975)

1 | 2 | 253 | POINT(3427927.775 5793243.925)

2 | 2 | 254 | POINT(3427927.825 5793243.925)

3 | 2 | 254 | POINT(3427927.875 5793243.925)

4 | 2 | 253 | POINT(3427927.925 5793243.925)

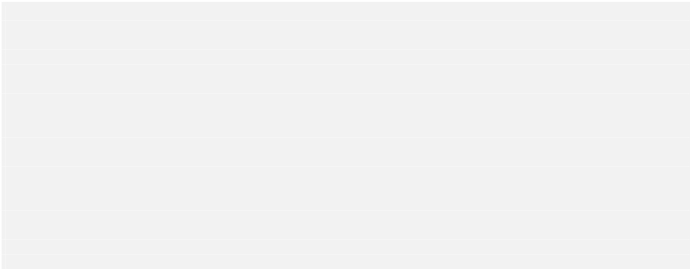
5 | 2 | 249 | POINT(3427927.975 5793243.925)

1 | 3 | 250 | POINT(3427927.775 5793243.875)

2 | 3 | 254 | POINT(3427927.825 5793243.875)

3 | 3 | 254 | POINT(3427927.875 5793243.875)

4 | 3 | 252 | POINT(3427927.925 5793243.875)



PostGIS 2.2.0dev 开发手册

418 / 686

5

1

2

3

4

5

1

2

3

4

5

|

|

|

|

|

|

|

|

|

|

|

3

4

4

4

4

4

5

5

5

5

5

|

|

|

|

|

|

|

|

|

|

|

249

251

253

254

254

253

252

250

254

254

254

|

|

|

|

|

|

|

|

|

|

|

POINT(3427927.975

POINT(3427927.775

POINT(3427927.825

POINT(3427927.875

POINT(3427927.925

POINT(3427927.975

POINT(3427927.775

POINT(3427927.825

POINT(3427927.875

POINT(3427927.925

POINT(3427927.975

5793243.875)

5793243.825)

5793243.825)

5793243.825)

5793243.825)

5793243.825)

5793243.775)

5793243.775)

5793243.775)

5793243.775)

5793243.775)

**相关请参考**

ST\_DumpAsPolygons, ST\_PixelAsPolygon, ST\_PixelAsPolygons, ST\_PixelAsPoint, ST\_PixelAsPoints, ST\_PixelAsCentroid

**9.6.7**

**ST\_Value**

ST\_Value — 返回给定波段的行和列的像元的值。波段如果没有指定是哪个，默认从1开始。如果exclude\_nodata\_value 值为false，那么所有包含NODATA值的像元都会返回value值，如果exclude\_nodata\_value没有指定，那么这个参数值将从栅格元数据中读取。

**用法**

double precision ST\_Value(raster rast, geometry pt, boolean exclude\_nodata\_value=true);

double precision ST\_Value(raster rast, integer bandnum, geometry pt, boolean exclude\_nodata\_value=true);

double precision ST\_Value(raster rast, integer columnx, integer rowy, boolean exclude\_nodata\_value=true);

double precision ST\_Value(raster rast, integer bandnum, integer columnx, integer rowy, boolean exclude\_nodata\_value=true);

**描述**

返回给定波段的行和列的像元的值。波段如果没有指定是哪个，默认从1开始。如果exclude\_nodata\_value 值为false，那么所有包含NODATA值的像元都会返回value值，如果这个值为true，那么只有不包含NODATA值的像素会应用本函数。

版本提升： 2.0.0 添加了参数exclude\_nodata\_value 支持

**样例**

-- get raster values at particular postgis geometry points

-- the srid of your geometry should be same as for your raster

SELECT rid, ST\_Value(rast, foo.pt\_geom) As b1pval, ST\_Value(rast, 2, foo.pt\_geom) As b2pval

FROM dummy\_rast CROSS JOIN (SELECT ST\_SetSRID(ST\_Point(3427927.77, 5793243.76), 0) As

pt\_geom) As foo

WHERE rid=2;

rid | b1pval | b2pval

-----+--------+--------

2 | 252 | 79

-- general fictitious example using a real table

SELECT rid, ST\_Value(rast, 3, sometable.geom) As b3pval

FROM sometable

WHERE ST\_Intersects(rast,sometable.geom);

←

PostGIS 2.2.0dev 开发手册

419 / 686

SELECT rid, ST\_Value(rast, 1, 1, 1) As b1pval,

ST\_Value(rast, 2, 1, 1) As b2pval, ST\_Value(rast, 3, 1, 1) As b3pval

FROM dummy\_rast

WHERE rid=2;

rid | b1pval | b2pval | b3pval

-----+--------+--------+--------

2 | 253 | 78 | 70

--- Get all values in bands 1,2,3 of each pixel --

SELECT x, y, ST\_Value(rast, 1, x, y) As b1val,

ST\_Value(rast, 2, x, y) As b2val, ST\_Value(rast, 3, x, y) As b3val

FROM dummy\_rast CROSS JOIN

generate\_series(1, 1000) As x CROSS JOIN generate\_series(1, 1000) As y

WHERE rid = 2 AND x <= ST\_Width(rast) AND y <= ST\_Height(rast);

x | y | b1val | b2val | b3val

---+---+-------+-------+-------

1 | 1 | 253 | 78 | 70

1 | 2 | 253 | 96 | 80

1 | 3 | 250 | 99 | 90

1 | 4 | 251 | 89 | 77

1 | 5 | 252 | 79 | 62

2 | 1 | 254 | 98 | 86

2 | 2 | 254 | 118 | 108

:

:

--- Get all values in bands 1,2,3 of each pixel same as above but returning the upper left

point point of each pixel --

SELECT ST\_AsText(ST\_SetSRID(

ST\_Point(ST\_UpperLeftX(rast) + ST\_ScaleX(rast)\*x,

ST\_UpperLeftY(rast) + ST\_ScaleY(rast)\*y),

ST\_SRID(rast))) As uplpt

, ST\_Value(rast, 1, x, y) As b1val,

ST\_Value(rast, 2, x, y) As b2val, ST\_Value(rast, 3, x, y) As b3val

FROM dummy\_rast CROSS JOIN

generate\_series(1,1000) As x CROSS JOIN generate\_series(1,1000) As y

WHERE rid = 2 AND x <= ST\_Width(rast) AND y <= ST\_Height(rast);

←

uplpt

| b1val | b2val | b3val

-----------------------------+-------+-------+-------

POINT(3427929.25 5793245.5) | 253 | 78 | 70

POINT(3427929.25 5793247) | 253 | 96 | 80

POINT(3427929.25 5793248.5) | 250 | 99 | 90

:

--- Get a polygon formed by union of all pixels

that fall in a particular value range and intersect particular polygon --

SELECT ST\_AsText(ST\_Union(pixpolyg)) As shadow

FROM (SELECT ST\_Translate(ST\_MakeEnvelope(

ST\_UpperLeftX(rast), ST\_UpperLeftY(rast),

ST\_UpperLeftX(rast) + ST\_ScaleX(rast),

ST\_UpperLeftY(rast) + ST\_ScaleY(rast), 0

), ST\_ScaleX(rast)\*x, ST\_ScaleY(rast)\*y

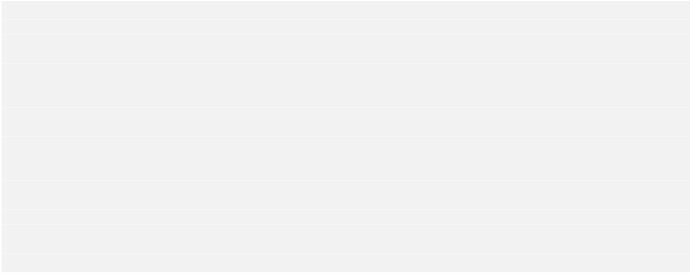
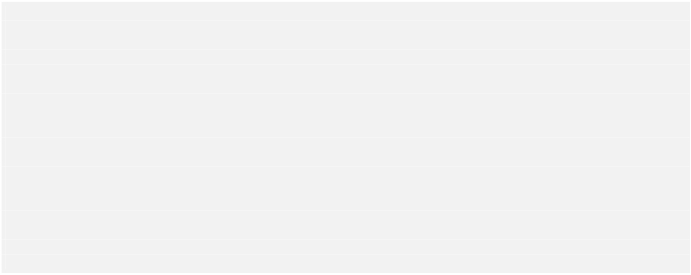
) As pixpolyg, ST\_Value(rast, 2, x, y) As b2val

FROM dummy\_rast CROSS JOIN

generate\_series(1,1000) As x CROSS JOIN generate\_series(1,1000) As y

WHERE rid = 2

AND x <= ST\_Width(rast) AND y <= ST\_Height(rast)) As foo



PostGIS 2.2.0dev 开发手册

420 / 686

WHERE

ST\_Intersects(

pixpolyg,

ST\_GeomFromText(’POLYGON((3427928 5793244,3427927.75 5793243.75,3427928

←

5793243.75,3427928 5793244))’,0)

) AND b2val != 254;

shadow

------------------------------------------------------------------------------------

MULTIPOLYGON(((3427928 5793243.9,3427928 5793243.85,3427927.95 5793243.85,3427927.95

5793243.9,

←

3427927.95 5793243.95,3427928 5793243.95,3427928.05 5793243.95,3427928.05

5793243.9,3427928 5793243.9)),((3427927.95 5793243.9,3427927.95 579324

←

3.85,3427927.9 5793243.85,3427927.85 5793243.85,3427927.85 5793243.9,3427927.9

5793243.9,3427927.9 5793243.95,

3427927.95 5793243.95,3427927.95 5793243.9)),((3427927.85 5793243.75,3427927.85

5793243.7,3427927.8 5793243.7,3427927.8 5793243.75

←

←

,3427927.8 5793243.8,3427927.8 5793243.85,3427927.85 5793243.85,3427927.85

5793243.8,3427927.85 5793243.75)),

((3427928.05 5793243.75,3427928.05 5793243.7,3427928 5793243.7,3427927.95

←

←

5793243.7,3427927.95 5793243.75,3427927.95 5793243.8,3427

927.95 5793243.85,3427928 5793243.85,3427928 5793243.8,3427928.05 5793243.8,

3427928.05 5793243.75)),((3427927.95 5793243.75,3427927.95 5793243.7,3427927.9

←

5793243.7,3427927.85 5793243.7,

3427927.85 5793243.75,3427927.85 5793243.8,3427927.85 5793243.85,3427927.9 5793243.85,

3427927.95 5793243.85,3427927.95 5793243.8,3427927.95 5793243.75)))

---

---

--

Checking all the pixels of a large raster tile can take a long time.

You can dramatically improve speed at some lose of precision by orders of magnitude

by sampling pixels using the step optional parameter of generate\_series.

--

--

This next example does the same as previous but by checking 1 for every 4 (2x2) pixels

and putting in the last checked

putting in the checked pixel as the value for subsequent 4

←

SELECT ST\_AsText(ST\_Union(pixpolyg)) As shadow

FROM (SELECT ST\_Translate(ST\_MakeEnvelope(

ST\_UpperLeftX(rast), ST\_UpperLeftY(rast),

ST\_UpperLeftX(rast) + ST\_ScaleX(rast)\*2,

ST\_UpperLeftY(rast) + ST\_ScaleY(rast)\*2, 0

), ST\_ScaleX(rast)\*x, ST\_ScaleY(rast)\*y

) As pixpolyg, ST\_Value(rast, 2, x, y) As b2val

FROM dummy\_rast CROSS JOIN

generate\_series(1,1000,2) As x CROSS JOIN generate\_series(1,1000,2) As y

WHERE rid = 2

AND x <= ST\_Width(rast)

WHERE

ST\_Intersects(

pixpolyg,

AND y <= ST\_Height(rast)

) As foo

ST\_GeomFromText(’POLYGON((3427928 5793244,3427927.75 5793243.75,3427928

←

5793243.75,3427928 5793244))’,0)

) AND b2val != 254;

shadow

------------------------------------------------------------------------------------

MULTIPOLYGON(((3427927.9 5793243.85,3427927.8 5793243.85,3427927.8 5793243.95,

3427927.9 5793243.95,3427928 5793243.95,3427928.1 5793243.95,3427928.1 5793243.85,3427928

5793243.85,3427927.9 5793243.85)),

←

((3427927.9 5793243.65,3427927.8 5793243.65,3427927.8 5793243.75,3427927.8

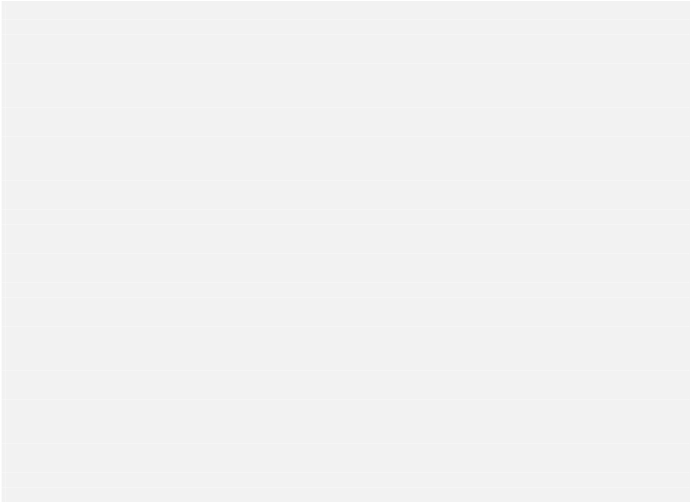
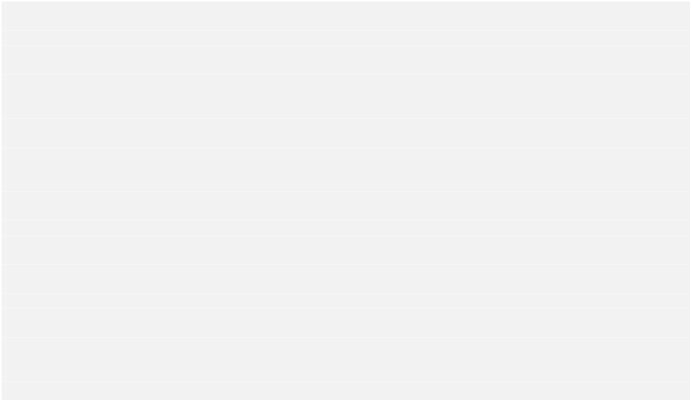
5793243.85,3427927.9 5793243.85,

←

3427928 5793243.85,3427928 5793243.75,3427928.1 5793243.75,3427928.1 5793243.65,3427928

5793243.65,3427927.9 5793243.65)))

←



PostGIS 2.2.0dev 开发手册

421 / 686

**相关请参考**

ST\_DumpAsPolygons, ST\_NumBands, ST\_PixelAsPolygon, ST\_ScaleX, , ST\_ScaleY, ST\_UpperLeftX, ST\_UpperLeftY, ST\_SRID,

ST\_AsText, , ST\_Point, ST\_MakeEnvelope, ST\_Intersects, ST\_Intersection

**9.6.8**

**ST\_NearestValue**

ST\_NearestValue — 返回给定波段的给定像元的最近的非NODATA的像元的值，给定像元可以通过行和列的位置或者栅格所在的空间参考系的几何坐标指定。

**用法**

double precision ST\_NearestValue(raster rast, integer bandnum, geometry pt, boolean exclude\_nodata\_value=true);

double precision ST\_NearestValue(raster rast, geometry pt, boolean exclude\_nodata\_value=true);

double precision ST\_NearestValue(raster rast, integer bandnum, integer columnx, integer rowy, boolean exclude\_nodata\_value=true);

double precision ST\_NearestValue(raster rast, integer columnx, integer rowy, boolean exclude\_nodata\_value=true);

**描述**

返回给定波段的给定像元的最近的非NODATA的像元的值，给定像元可以通过行和列的位置或者栅格所在的空间参考系的几何坐标指定。如果指定的像元是NODATA的，那么这个函数会返回找到离改点最近的像元的值。波段如果没有指定是哪个，默认从1开始。如果exclude\_nodata\_value 值为false，那么所有包含NODATA值的像元都会返回value值，如果exclude\_nodata\_value没有指定，那么这个参数值将从栅格元数据中读取。

可用版本：2.1.0

**注意**

ST\_NearestValue 是函数ST\_Value的简易替换

**样例**

-- pixel 2x2 has value

SELECT

ST\_Value(rast, 2, 2) AS value,

ST\_NearestValue(rast, 2, 2) AS nearestvalue

FROM (

SELECT

ST\_SetValue(

ST\_SetValue(

ST\_SetValue(

ST\_SetValue(

ST\_SetValue(

ST\_AddBand(

ST\_MakeEmptyRaster(5, 5, -2, 2, 1, -1, 0, 0, 0),

’8BUI’::text, 1, 0

),

1, 1, 0.



PostGIS 2.2.0dev 开发手册

422 / 686

),

2, 3, 0.

),

3, 5, 0.

),

4, 2, 0.

),

5, 4, 0.

) AS rast

) AS foo

value | nearestvalue

-------+--------------

1 | 1

-- pixel 2x3 is NODATA

SELECT

ST\_Value(rast, 2, 3) AS value,

ST\_NearestValue(rast, 2, 3) AS nearestvalue

FROM (

SELECT

ST\_SetValue(

ST\_SetValue(

ST\_SetValue(

ST\_SetValue(

ST\_SetValue(

ST\_AddBand(

ST\_MakeEmptyRaster(5, 5, -2, 2, 1, -1, 0, 0, 0),

’8BUI’::text, 1, 0

),

1, 1, 0.

),

2, 3, 0.

),

3, 5, 0.

),

4, 2, 0.

),

5, 4, 0.

) AS rast

) AS foo

value | nearestvalue

-------+--------------

| 1

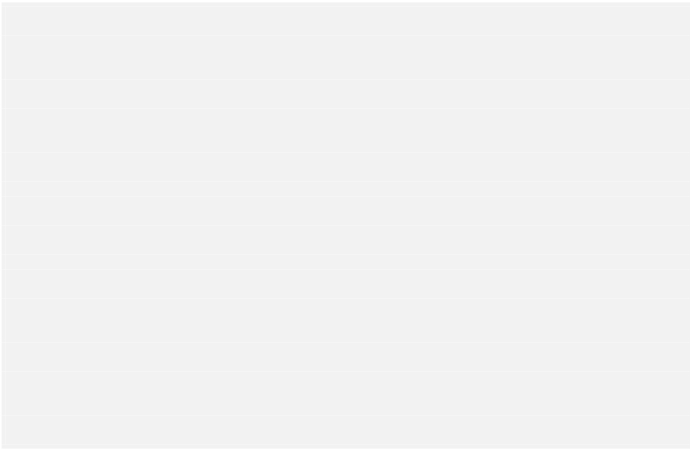
**相关请参考**

ST\_Neighborhood, ST\_Value

**9.6.9**

**ST\_Neighborhood**

ST\_Neighborhood —返回一个2维双精度的不包含NODATA值的像元值数组，像元是在给定波段像元周围。像元坐标用行列位置指定或几何点指定，参考系以栅格所处参考系为准。



PostGIS 2.2.0dev 开发手册

423 / 686

**用法**

double precision[][] ST\_Neighborhood(raster rast, integer bandnum, integer columnX, integer rowY, integer distanceX, integer

distanceY, boolean exclude\_nodata\_value=true);

double precision[][] ST\_Neighborhood(raster rast, integer columnX, integer rowY, integer distanceX, integer distanceY, boolean

exclude\_nodata\_value=true);

double precision[][] ST\_Neighborhood(raster rast, integer bandnum, geometry pt, integer distanceX, integer distanceY, boolean

exclude\_nodata\_value=true);

double precision[][] ST\_Neighborhood(raster rast, geometry pt, integer distanceX, integer distanceY, boolean exclude\_nodata\_value=true

**描述**

返回一个2维双精度的不包含NODATA值的像元值数组，像元是在给定波段像元周围。像元坐标用行列位置指定或几何点指定，参考系以栅格所处参考系为准。参数distanceX 和distanceY 定义了指定像元在x轴和y轴方向上的像元个数。例如，我想知道我感兴趣的像元在x轴方向上3个像素单位，y轴方向上2个像素单位以内的所有像素值。返回2维数组的中心点，就是提供的参数点（我们感兴趣的的点）。

波段如果没有指定是哪个，默认从1开始。如果exclude\_nodata\_value 值为false，那么所有包含NODATA值的像元都会返回value值，如果exclude\_nodata\_value没有指定，那么这个参数值将从栅格元数据中读取。

**注意**

返回2维数组的沿着每个轴方向的元素个数是2 \* (distanceX|distanceY) + 1，因此对于distanceX和 distanceY都是1的情况，返回的数组将是3x3的

**注意**

输出的2维数组可以用于任意的内置的栅格处理函数，例如ST\_Min4ma, ST\_Sum4ma,ST\_Mean4ma.

可用版本：2.1.0

**样例**

-- pixel 2x2 has value

SELECT

ST\_Neighborhood(rast, 2, 2, 1, 1)

FROM (

SELECT

ST\_SetValues(

ST\_AddBand(

ST\_MakeEmptyRaster(5, 5, -2, 2, 1, -1, 0, 0, 0),

’8BUI’::text, 1, 0

),

1, 1, 1, ARRAY[

[0, 1, 1, 1, 1],

[1, 1, 1, 0, 1],

[1, 0, 1, 1, 1],

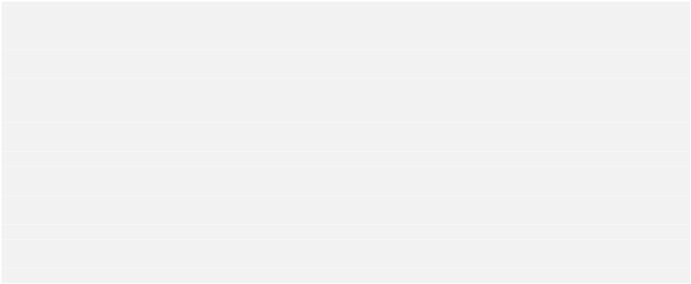
[1, 1, 1, 1, 0],

[1, 1, 0, 1, 1]

]::double precision[],

1

) AS rast



PostGIS 2.2.0dev 开发手册

424 / 686

) AS foo

st\_neighborhood

---------------------------------

{{NULL,1,1},{1,1,NULL},{1,1,1}}

-- pixel 2x3 is NODATA

SELECT

ST\_Neighborhood(rast, 2, 3, 1, 1)

FROM (

SELECT

ST\_SetValues(

ST\_AddBand(

ST\_MakeEmptyRaster(5, 5, -2, 2, 1, -1, 0, 0, 0),

’8BUI’::text, 1, 0

),

1, 1, 1, ARRAY[

[0, 1, 1, 1, 1],

[1, 1, 1, 0, 1],

[1, 0, 1, 1, 1],

[1, 1, 1, 1, 0],

[1, 1, 0, 1, 1]

]::double precision[],

1

) AS rast

) AS foo

st\_neighborhood

------------------------------

{{1,1,1},{1,NULL,1},{1,1,1}}

-- pixel 3x3 has value

-- exclude\_nodata\_value = FALSE

SELECT

ST\_Neighborhood(rast, 3, 3, 1, 1, false)

FROM (

ST\_SetValues(

ST\_AddBand(

ST\_MakeEmptyRaster(5, 5, -2, 2, 1, -1, 0, 0, 0),

’8BUI’::text, 1, 0

),

1, 1, 1, ARRAY[

[0, 1, 1, 1, 1],

[1, 1, 1, 0, 1],

[1, 0, 1, 1, 1],

[1, 1, 1, 1, 0],

[1, 1, 0, 1, 1]

]::double precision[],

1

) AS rast

) AS foo

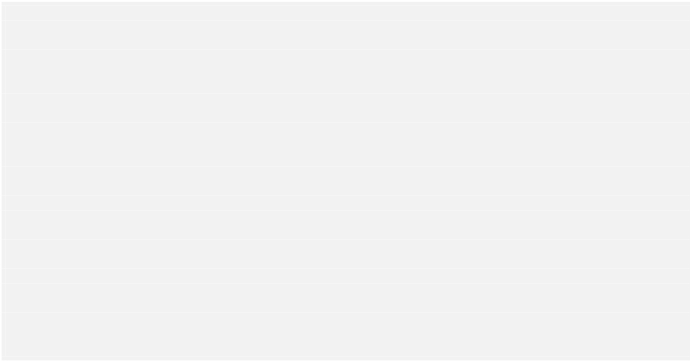
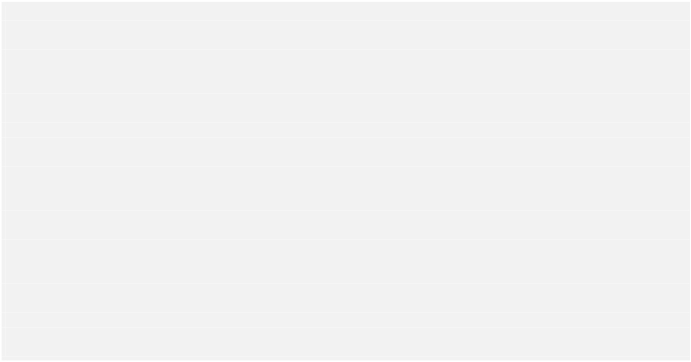
st\_neighborhood

---------------------------

{{1,0,1},{1,1,1},{0,1,1}}

**相关请参考**

ST\_NearestValue, ST\_Min4ma, ST\_Max4ma, ST\_Sum4ma, ST\_Mean4ma, ST\_Range4ma, ST\_Distinct4ma, ST\_StdDev4ma



PostGIS 2.2.0dev 开发手册

425 / 686

**9.6.10**

**ST\_SetValue**

ST\_SetValue — 根据给定波段的给定像元，设定其值，返回一个新的修改后的栅格。

波段如果没有指定是哪个，默认从1开始。

**用法**

raster ST\_SetValue(raster rast, integer bandnum, geometry geom, double precision newvalue);

raster ST\_SetValue(raster rast, geometry geom, double precision newvalue);

raster ST\_SetValue(raster rast, integer bandnum, integer columnx, integer rowy, double precision newvalue);

raster ST\_SetValue(raster rast, integer columnx, integer rowy, double precision newvalue);

**描述**

根据给定波段的给定像元，设定其值，返回一个新的修改后的栅格。如果没有指定波段，默认是波段1

版本提升： 2.1.0 ST\_SetValue() 现在支持包括点在内的任意几何类型对象。几何参数geom是一个ST\_SetValues()变量geomval[]的包装。

**样例**

-- Geometry example

SELECT (foo.geomval).val, ST\_AsText(ST\_Union((foo.geomval).geom))

FROM (SELECT ST\_DumpAsPolygons(

ST\_SetValue(rast,1,

ST\_Point(3427927.75, 5793243.95),

50)

) As geomval

FROM dummy\_rast

where rid = 2) As foo

WHERE (foo.geomval).val < 250

GROUP BY (foo.geomval).val;

val |

st\_astext

-----+-------------------------------------------------------------------

50 | POLYGON((3427927.75 5793244,3427927.75 5793243.95,3427927.8 579324 ...

249 | POLYGON((3427927.95 5793243.95,3427927.95 5793243.85,3427928 57932 ...

-- Store the changed raster --

UPDATE dummy\_rast SET rast = ST\_SetValue(rast,1, ST\_Point(3427927.75, 5793243.95),100)

WHERE rid = 2 ;

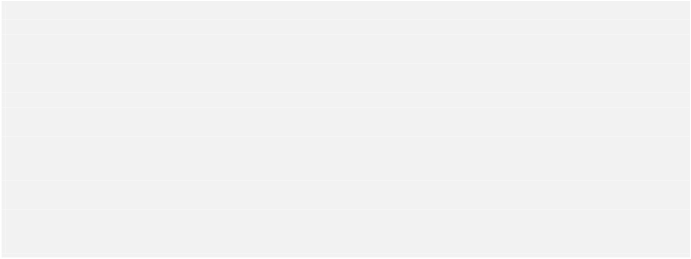
**相关请参考**

ST\_Value, ST\_DumpAsPolygons

**9.6.11**

**ST\_SetValues**

ST\_SetValues —根据给定的栅格波段值，设置新值，返回被设定新波段值的栅格



PostGIS 2.2.0dev 开发手册

426 / 686

**用法**

raster ST\_SetValues(raster rast, integer nband, integer columnx, integer rowy, double precision[][] newvalueset, boolean[][]

noset=NULL, boolean keepnodata=FALSE);

raster ST\_SetValues(raster rast, integer nband, integer columnx, integer rowy, double precision[][] newvalueset, double precision

nosetvalue, boolean keepnodata=FALSE);

raster ST\_SetValues(raster rast, integer nband, integer columnx, integer rowy, integer width, integer height, double precision

newvalue, boolean keepnodata=FALSE);

raster ST\_SetValues(raster rast, integer columnx, integer rowy, integer width, integer height, double precision newvalue, boolean

keepnodata=FALSE);

raster ST\_SetValues(raster rast, integer nband, geomval[] geomvalset, boolean keepnodata=FALSE);

**描述**

根据给定的栅格波段值，设置新值，返回被设定新波段值的栅格。

如果参数keepnodata 值是 TRUE, 哪些像元值是NODATA的将不会设置成对应的newvalueset变量值。

对于函数变体1，指定被修改值的像素是由参数columnx, rowy 像素坐标和数组变量newvalueset的维度决定的。

参数noset 可以防止值在newvalueset 内的像素被设置值。 (因为PostgreSQL 不允许ragged/jagged类型数组). 参考样例1.

函数变体2与函数变体1类似，但是使用的是一个简单双精度nosetvalue数组代替了逻辑值数组noset。若nosetvalue值在newvalueset数组中，则会被跳过不修改。 with the nosetvalue value with be skipped. 参考样例2.

对于函数变体3，被设定值的像元是由参数columnx, rowy 像元坐标, width 和 height参数决定的.参考样例3.

函数变体4和函数变体3基本相同，区别是它假定栅格波段的第一个像素会被设置值。

对于函数变体5，数组变量 geomval用于指定被修改的像素范围。如果数组内的所有几何对象都是POINT或MULTIPOINT类型，这个函数会使用快捷办法，也就是直接使用每个点的经纬度来进行设置。否则这些几何对象会被转换成栅格，然后再遍历一次按照栅格的方法处理。

参考样例5.

可用版本：2.1.0

**样例: Variant 1**

/\*

The ST\_SetValues() does the following...

+ - + - + - + + - + - +

| 1 | 1 | 1 | | 1 | 1 |

+ - + - + - + + - + - +

| 1 | 1 | 1 | => | 1 | 9 |

+ - + - + - + + - + - +

| 1 | 1 | 1 | | 1 | 9 |

+ - + - + - + + - + - +

\*/

SELECT

(poly).x,

(poly).y,

(poly).val

FROM (

SELECT

ST\_PixelAsPolygons(

ST\_SetValues(

ST\_AddBand(

-

1

-

9

-

9

-

+

|

+

|

+

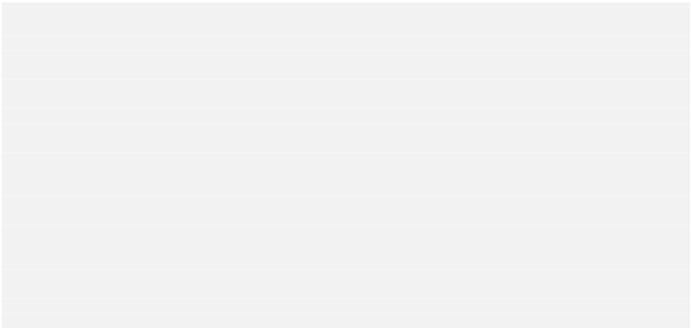
|

+

ST\_MakeEmptyRaster(3, 3,

1, ’8BUI’, 1, 0

0, 0, 1, -1, 0, 0, 0),



PostGIS 2.2.0dev 开发手册

427 / 686

),

1, 2, 2, ARRAY[[9, 9], [9, 9]]::double precision[][]

)

) AS poly

) foo

ORDER BY 1, 2;

x | y | val

---+---+-----

1 | 1 | 1

1 | 2 | 1

1 | 3 | 1

2 | 1 | 1

2 | 2 | 9

2 | 3 | 9

3 | 1 | 1

3 | 2 | 9

3 | 3 | 9

/\*

The ST\_SetValues() does the following...

+ - + - + - + + - + - + - +

| 1 | 1 | 1 | | 9 | 9 | 9 |

+ - + - + - + + - + - + - +

| 1 | 1 | 1 | => | 9 | | 9 |

+ - + - + - + + - + - + - +

| 1 | 1 | 1 | | 9 | 9 | 9 |

+ - + - + - + + - + - + - +

\*/

SELECT

(poly).x,

(poly).y,

(poly).val

FROM (

SELECT

ST\_PixelAsPolygons(

ST\_SetValues(

ST\_AddBand(

ST\_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0),

1, ’8BUI’, 1, 0

),

1, 1, 1, ARRAY[[9, 9, 9], [9, NULL, 9], [9, 9, 9]]::double precision[][]

)

) AS poly

) foo

ORDER BY 1, 2;

x | y | val

---+---+-----

1 | 1 | 9

1 | 2 | 9

1 | 3 | 9

2 | 1 | 9

2 | 2 |

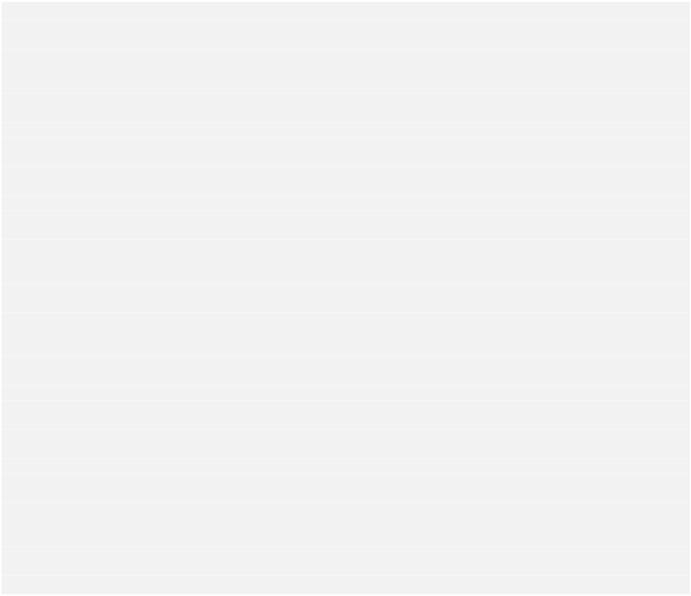
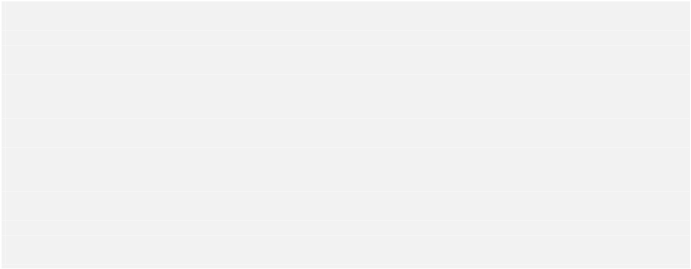
2 | 3 | 9

3 | 1 | 9

3 | 2 | 9

3 | 3 | 9

/\*



PostGIS 2.2.0dev 开发手册

428 / 686

The ST\_SetValues() does the following...

+ - + - + - + + - + - + - +

| 1 | 1 | 1 | | 9 | 9 | 9 |

+ - + - + - + + - + - + - +

| 1 | 1 | 1 | => | 1 | | 9 |

+ - + - + - + + - + - + - +

| 1 | 1 | 1 | | 9 | 9 | 9 |

+ - + - + - + + - + - + - +

\*/

SELECT

(poly).x,

(poly).y,

(poly).val

FROM (

SELECT

ST\_PixelAsPolygons(

ST\_SetValues(

ST\_AddBand(

ST\_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0),

1, ’8BUI’, 1, 0

),

1, 1, 1,

ARRAY[[9, 9, 9], [9, NULL, 9], [9, 9, 9]]::double precision[][],

ARRAY[[false], [true]]::boolean[][]

)

) AS poly

) foo

ORDER BY 1, 2;

x | y | val

---+---+-----

1 | 1 | 9

1 | 2 | 1

1 | 3 | 9

2 | 1 | 9

2 | 2 |

2 | 3 | 9

3 | 1 | 9

3 | 2 | 9

3 | 3 | 9

/\*

The ST\_SetValues() does the following...

+ - + - + - +

| | 1 | 1 |

+ - + - + - +

| 1 | 1 | 1 | =>

+ - + - + - +

| 1 | 1 | 1 |

+ - + - + - +

+

|

+

|

+

|

+

- +

|

- +

1 |

- +

9 |

- +

- + - +

9 | 9 |

- + - +

| 9 |

- + - +

9 | 9 |

- + - +

\*/

SELECT

(poly).x,

(poly).y,

(poly).val

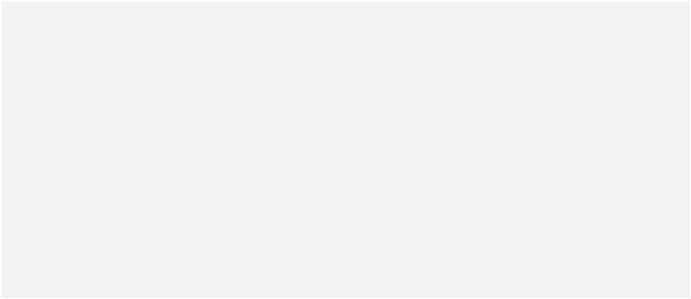
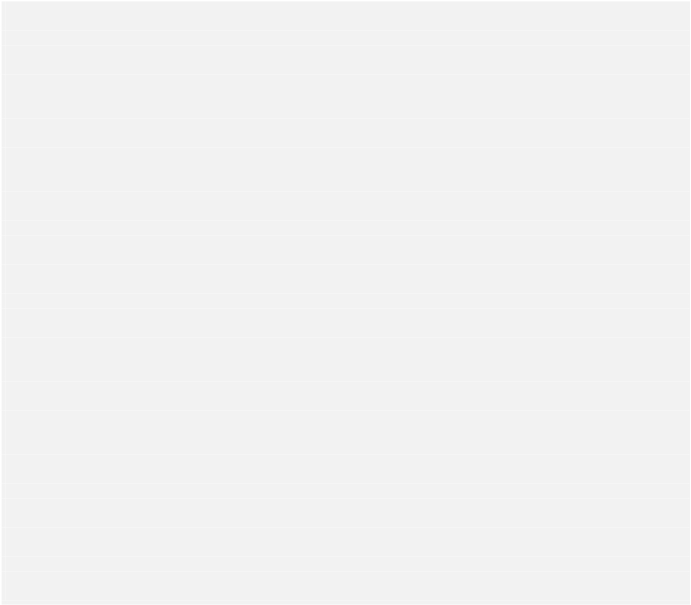
FROM (

SELECT

ST\_PixelAsPolygons(

ST\_SetValues(

ST\_SetValue(



PostGIS 2.2.0dev 开发手册

429 / 686

ST\_AddBand(

ST\_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0),

1, ’8BUI’, 1, 0

),

1, 1, 1, NULL

),

1, 1, 1,

ARRAY[[9, 9, 9], [9, NULL, 9], [9, 9, 9]]::double precision[][],

ARRAY[[false], [true]]::boolean[][],

TRUE

)

) AS poly

) foo

ORDER BY 1, 2;

x | y | val

---+---+-----

1 | 1 |

1 | 2 | 1

1 | 3 | 9

2 | 1 | 9

2 | 2 |

2 | 3 | 9

3 | 1 | 9

3 | 2 | 9

3 | 3 | 9

**样例: Variant 2**

/\*

The ST\_SetValues() does the following...

+ - + - + - + + - + - + - +

| 1 | 1 | 1 | | 1 | 1 | 1 |

+ - + - + - + + - + - + - +

| 1 | 1 | 1 | => | 1 | 9 | 9 |

+ - + - + - + + - + - + - +

| 1 | 1 | 1 | | 1 | 9 | 9 |

+ - + - + - + + - + - + - +

\*/

SELECT

(poly).x,

(poly).y,

(poly).val

FROM (

SELECT

ST\_PixelAsPolygons(

ST\_SetValues(

ST\_AddBand(

ST\_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0),

1, ’8BUI’, 1, 0

),

1, 1, 1, ARRAY[[-1, -1, -1], [-1, 9, 9], [-1, 9, 9]]::double precision[][], -1

)

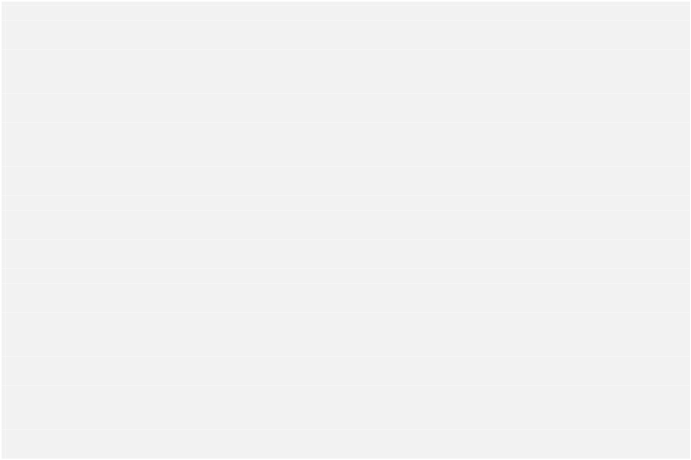
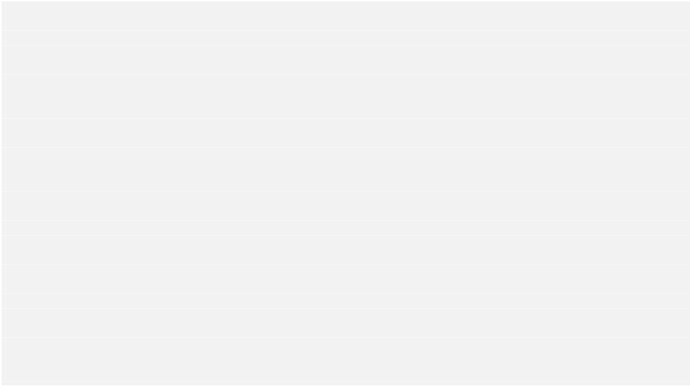
) AS poly

) foo

ORDER BY 1, 2;

x | y | val

---+---+-----



PostGIS 2.2.0dev 开发手册

430 / 686

1

1

1

2

2

2

3

3

3

|

|

|

|

|

|

|

|

|

1

2

3

1

2

3

1

2

3

|

|

|

|

|

|

|

|

|

1

1

1

1

9

9

1

9

9

/\*

This example is like the previous one.

Instead of nosetvalue = -1, nosetvalue = NULL

The ST\_SetValues() does the following...

+ - + - + - + + - + - + - +

| 1 | 1 | 1 | | 1 | 1 | 1 |

+ - + - + - + + - + - + - +

| 1 | 1 | 1 | => | 1 | 9 | 9 |

+ - + - + - + + - + - + - +

| 1 | 1 | 1 | | 1 | 9 | 9 |

+ - + - + - + + - + - + - +

\*/

SELECT

(poly).x,

(poly).y,

(poly).val

FROM (

SELECT

ST\_PixelAsPolygons(

ST\_SetValues(

ST\_AddBand(

ST\_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0),

1, ’8BUI’, 1, 0

),

1, 1, 1, ARRAY[[NULL, NULL, NULL], [NULL, 9, 9], [NULL, 9, 9]]::double precision[][], ←

NULL::double precision

)

) AS poly

) foo

ORDER BY 1, 2;

x | y | val

---+---+-----

1 | 1 | 1

1 | 2 | 1

1 | 3 | 1

2 | 1 | 1

2 | 2 | 9

2 | 3 | 9

3 | 1 | 1

3 | 2 | 9

3 | 3 | 9

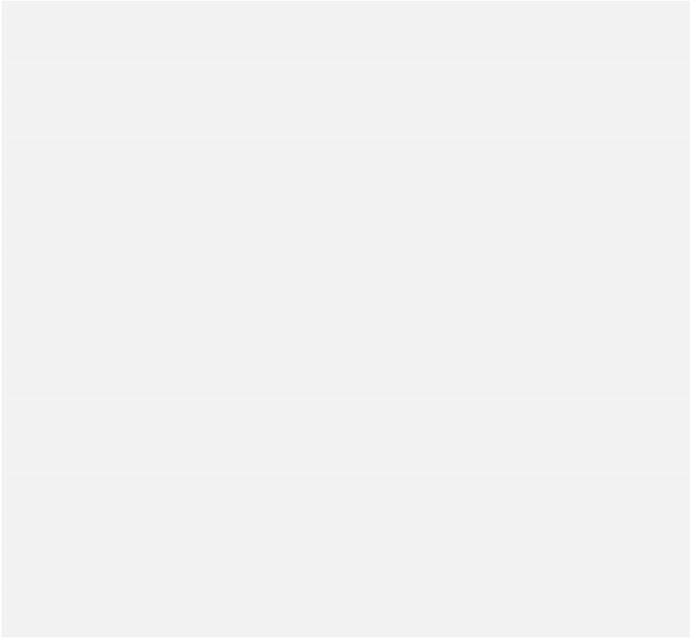
**样例: Variant 3**

/\*

The ST\_SetValues() does the following...

+ - + - + - +

+ - + - + - +



PostGIS 2.2.0dev 开发手册

431 / 686

| 1 | 1 | 1 | | 1 | 1 |

+ - + - + - + + - + - +

| 1 | 1 | 1 | => | 1 | 9 |

+ - + - + - + + - + - +

| 1 | 1 | 1 | | 1 | 9 |

+ - + - + - + + - + - +

\*/

SELECT

(poly).x,

(poly).y,

(poly).val

FROM (

SELECT

ST\_PixelAsPolygons(

ST\_SetValues(

ST\_AddBand(

1

-

9

-

9

-

|

+

|

+

|

+

ST\_MakeEmptyRaster(3, 3,

0, 0, 1, -1, 0, 0, 0),

1, ’8BUI’, 1, 0

),

1, 2, 2, 2, 2, 9

)

) AS poly

) foo

ORDER BY 1, 2;

x | y | val

---+---+-----

1 | 1 | 1

1 | 2 | 1

1 | 3 | 1

2 | 1 | 1

2 | 2 | 9

2 | 3 | 9

3 | 1 | 1

3 | 2 | 9

3 | 3 | 9

/\*

The ST\_SetValues() does the following...

+ - + - + - + + - + - + -

| 1 | 1 | 1 | | 1 | 1 | 1

+ - + - + - + + - + - + -

| 1 | | 1 | => | 1 | | 9

+ - + - + - + + - + - + -

| 1 | 1 | 1 | | 1 | 9 | 9

+ - + - + - + + - + - + -

\*/

SELECT

(poly).x,

(poly).y,

(poly).val

FROM (

SELECT

ST\_PixelAsPolygons(

ST\_SetValues(

ST\_SetValue(

ST\_AddBand(

ST\_MakeEmptyRaster(3, 3,

1, ’8BUI’, 1, 0

),

1, 2, 2, NULL

+

|

+

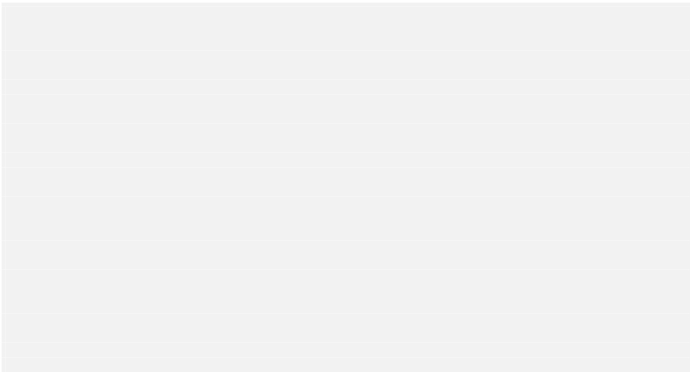
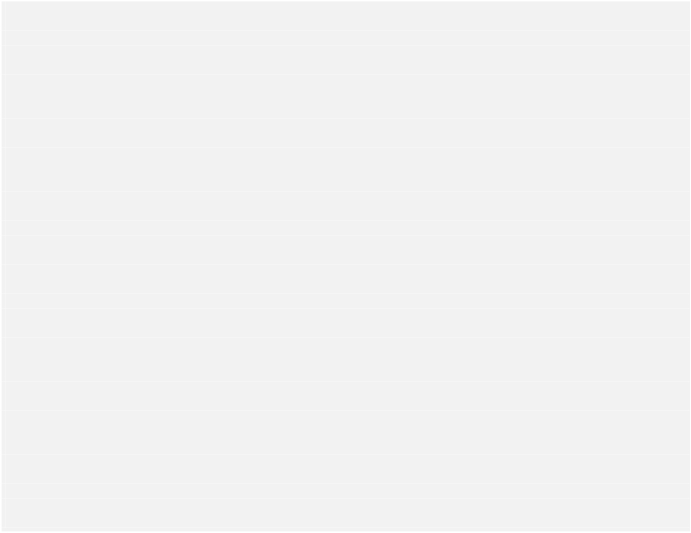
|

+

|

+

0, 0, 1, -1, 0, 0, 0),



PostGIS 2.2.0dev 开发手册

432 / 686

),

1, 2, 2, 2, 2, 9, TRUE

)

) AS poly

) foo

ORDER BY 1, 2;

x | y | val

---+---+-----

1 | 1 | 1

1 | 2 | 1

1 | 3 | 1

2 | 1 | 1

2 | 2 |

2 | 3 | 9

3 | 1 | 1

3 | 2 | 9

3 | 3 | 9

**样例: Variant 5**

WITH foo AS (

SELECT 1 AS rid, ST\_AddBand(ST\_MakeEmptyRaster(5, 5, 0, 0, 1, -1, 0, 0, 0), 1, ’8BUI’, 0, ←

0) AS rast

), bar AS (

SELECT 1 AS gid, ’SRID=0;POINT(2.5 -2.5)’::geometry geom UNION ALL

SELECT 2 AS gid, ’SRID=0;POLYGON((1 -1, 4 -1, 4 -4, 1 -4, 1 -1))’::geometry geom UNION

←

ALL

SELECT 3 AS gid, ’SRID=0;POLYGON((0 0, 5 0, 5 -1, 1 -1, 1 -4, 0 -4, 0 0))’::geometry geom ←

UNION ALL

SELECT 4 AS gid, ’SRID=0;MULTIPOINT(0 0, 4 4, 4 -4)’::geometry

)

SELECT

rid, gid, ST\_DumpValues(ST\_SetValue(rast, 1, geom, gid))

FROM foo t1

CROSS JOIN bar t2

ORDER BY rid, gid;

rid | gid |

st\_dumpvalues

-----+-----+-----------------------------------------------------------------------------------------

1 |

1 | (1,"{{NULL,NULL,NULL,NULL,NULL},{NULL,NULL,NULL,NULL,NULL},{NULL,NULL,1,NULL, ←

NULL},{NULL,NULL,NULL,NULL,NULL},{NULL,NULL,NULL,NULL,NULL}}")

1 | 2 | (1,"{{NULL,NULL,NULL,NULL,NULL},{NULL,2,2,2,NULL},{NULL,2,2,2,NULL},{NULL ←

,2,2,2,NULL},{NULL,NULL,NULL,NULL,NULL}}")

1 | 3 | (1,"{{3,3,3,3,3},{3,NULL,NULL,NULL,NULL},{3,NULL,NULL,NULL,NULL},{3,NULL,NULL, ←

NULL,NULL},{NULL,NULL,NULL,NULL,NULL}}")

1 | 4 | (1,"{{4,NULL,NULL,NULL,NULL},{NULL,NULL,NULL,NULL,NULL},{NULL,NULL,NULL,NULL, ←

NULL},{NULL,NULL,NULL,NULL,NULL},{NULL,NULL,NULL,NULL,4}}")

(4 rows)

The following shows that geomvals later in the array can overwrite prior geomvals

WITH foo AS (

SELECT 1 AS rid, ST\_AddBand(ST\_MakeEmptyRaster(5, 5, 0, 0, 1, -1, 0, 0, 0), 1, ’8BUI’, 0, ←

0) AS rast

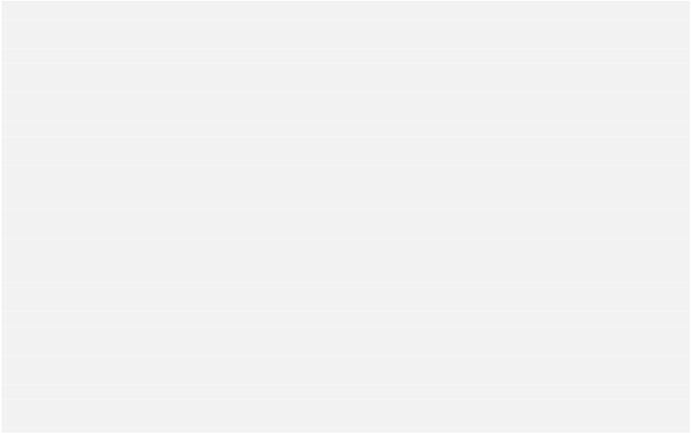
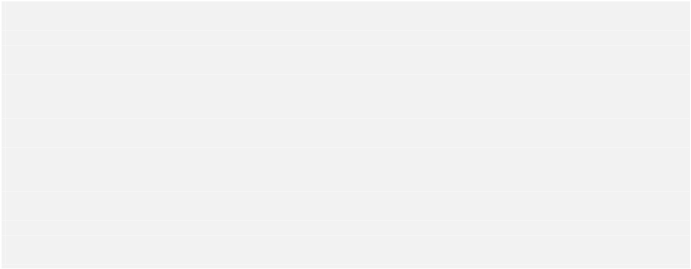
), bar AS (

SELECT 1 AS gid, ’SRID=0;POINT(2.5 -2.5)’::geometry geom UNION ALL

SELECT 2 AS gid, ’SRID=0;POLYGON((1 -1, 4 -1, 4 -4, 1 -4, 1 -1))’::geometry geom UNION

ALL

←



PostGIS 2.2.0dev 开发手册

433 / 686

SELECT 3 AS gid, ’SRID=0;POLYGON((0 0, 5 0, 5 -1, 1 -1, 1 -4, 0 -4, 0 0))’::geometry geom ←

UNION ALL

SELECT 4 AS gid, ’SRID=0;MULTIPOINT(0 0, 4 4, 4 -4)’::geometry

)

SELECT

t1.rid, t2.gid, t3.gid, ST\_DumpValues(ST\_SetValues(rast, 1, ARRAY[ROW(t2.geom, t2.gid),

ROW(t3.geom, t3.gid)]::geomval[]))

FROM foo t1

CROSS JOIN bar t2

CROSS JOIN bar t3

WHERE t2.gid = 1

AND t3.gid = 2

ORDER BY t1.rid, t2.gid, t3.gid;

←

rid | gid | gid |

st\_dumpvalues

-----+-----+-----+-----------------------------------------------------------------------------------

1 |

1 | 2 | (1,"{{NULL,NULL,NULL,NULL,NULL},{NULL,2,2,2,NULL},{NULL,2,2,2,NULL},{ ←

NULL,2,2,2,NULL},{NULL,NULL,NULL,NULL,NULL}}")

(1 row)

This example is the opposite of the prior example

WITH foo AS (

SELECT 1 AS rid, ST\_AddBand(ST\_MakeEmptyRaster(5, 5, 0, 0, 1, -1, 0, 0, 0), 1, ’8BUI’, 0, ←

0) AS rast

), bar AS (

SELECT 1 AS gid, ’SRID=0;POINT(2.5 -2.5)’::geometry geom UNION ALL

SELECT 2 AS gid, ’SRID=0;POLYGON((1 -1, 4 -1, 4 -4, 1 -4, 1 -1))’::geometry geom UNION

←

ALL

SELECT 3 AS gid, ’SRID=0;POLYGON((0 0, 5 0, 5 -1, 1 -1, 1 -4, 0 -4, 0 0))’::geometry geom ←

UNION ALL

SELECT 4 AS gid, ’SRID=0;MULTIPOINT(0 0, 4 4, 4 -4)’::geometry

)

SELECT

t1.rid, t2.gid, t3.gid, ST\_DumpValues(ST\_SetValues(rast, 1, ARRAY[ROW(t2.geom, t2.gid),

ROW(t3.geom, t3.gid)]::geomval[]))

FROM foo t1

CROSS JOIN bar t2

CROSS JOIN bar t3

WHERE t2.gid = 2

AND t3.gid = 1

ORDER BY t1.rid, t2.gid, t3.gid;

←

rid | gid | gid |

st\_dumpvalues

-----+-----+-----+-----------------------------------------------------------------------------------

1 |

2 | 1 | (1,"{{NULL,NULL,NULL,NULL,NULL},{NULL,2,2,2,NULL},{NULL,2,1,2,NULL},{ ←

NULL,2,2,2,NULL},{NULL,NULL,NULL,NULL,NULL}}")

(1 row)

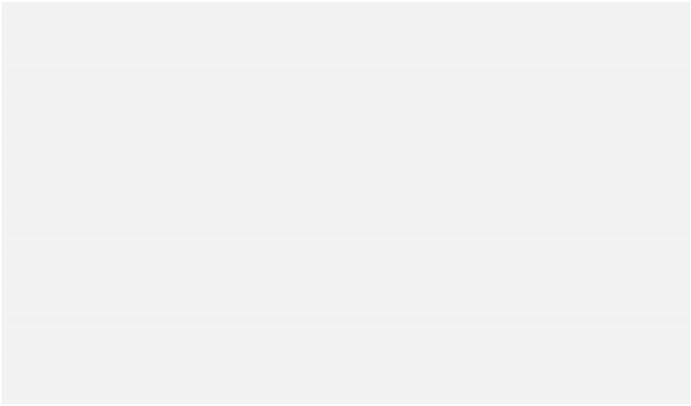
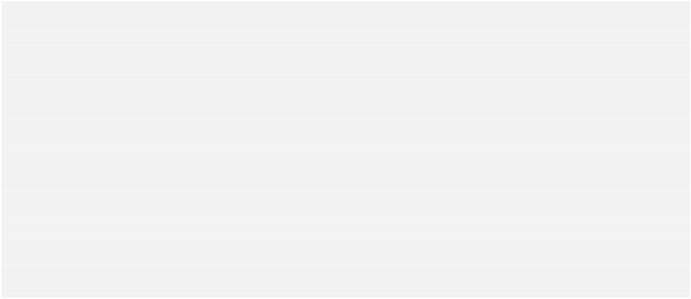
**相关请参考**

ST\_Value, ST\_SetValue, ST\_PixelAsPolygons

**9.6.12**

**ST\_DumpValues**

ST\_DumpValues — 以2维数组形式，返回指定波段的值



PostGIS 2.2.0dev 开发手册

434 / 686

**用法**

setof record ST\_DumpValues( raster rast , integer[] nband , boolean exclude\_nodata\_value=true );

double precision[][] ST\_DumpValues( raster rast , integer nband , boolean exclude\_nodata\_value=true );

**描述**

以2维数组形式，返回指定波段的值。如果参数nband是NULL，或者没有提供，那么所有的栅格波段都不会被处理

可用版本：2.1.0

**样例**

WITH foo AS (

SELECT ST\_AddBand(ST\_AddBand(ST\_AddBand(ST\_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0),

1, ’8BUI’, 1, 0), 2, ’32BF’, 3, -9999), 3, ’16BSI’, 0, 0) AS rast

)

SELECT

(ST\_DumpValues(rast)).\*

FROM foo;

←

nband |

valarray

-------+------------------------------------------------------

1 | {{1,1,1},{1,1,1},{1,1,1}}

2 | {{3,3,3},{3,3,3},{3,3,3}}

3 | {{NULL,NULL,NULL},{NULL,NULL,NULL},{NULL,NULL,NULL}}

(3 rows)

WITH foo AS (

SELECT ST\_AddBand(ST\_AddBand(ST\_AddBand(ST\_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0),

1, ’8BUI’, 1, 0), 2, ’32BF’, 3, -9999), 3, ’16BSI’, 0, 0) AS rast

)

SELECT

(ST\_DumpValues(rast, ARRAY[3, 1])).\*

FROM foo;

←

nband |

valarray

-------+------------------------------------------------------

3 | {{NULL,NULL,NULL},{NULL,NULL,NULL},{NULL,NULL,NULL}}

1 | {{1,1,1},{1,1,1},{1,1,1}}

(2 rows)

**相关请参考**

ST\_Value, ST\_SetValue, ST\_SetValues

**9.6.13**

**ST\_PixelOfValue**

ST\_PixelOfValue — 根据给定的输入值，返回所有匹配的像素的columnx，rowy坐标记录

**用法**

setof record ST\_PixelOfValue( raster rast , integer nband , double precision[] search , boolean exclude\_nodata\_value=true );

setof record ST\_PixelOfValue( raster rast , double precision[] search , boolean exclude\_nodata\_value=true );

setof record ST\_PixelOfValue( raster rast , integer nband , double precision search , boolean exclude\_nodata\_value=true );

setof record ST\_PixelOfValue( raster rast , double precision search , boolean exclude\_nodata\_value=true );

PostGIS 2.2.0dev 开发手册

435 / 686

**描述**

根据给定的输入值，返回所有匹配的像素的columnx，rowy坐标记录，如果没有指定波段，那么默认为波段1

可用版本：2.1.0

**样例**

SELECT

(pixels).\*

FROM (

SELECT

ST\_PixelOfValue(

ST\_SetValue(

ST\_SetValue(

ST\_SetValue(

ST\_SetValue(

ST\_SetValue(

ST\_AddBand(

ST\_MakeEmptyRaster(5, 5, -2, 2, 1, -1, 0, 0, 0),

’8BUI’::text, 1, 0

),

1, 1, 0

),

2, 3, 0

),

3, 5, 0

),

4, 2, 0

),

5, 4, 255

)

, 1, ARRAY[1, 255]) AS pixels

) AS foo

val | x | y

-----+---+---

1 | 1 | 2

1 | 1 | 3

1 | 1 | 4

1 | 1 | 5

1 | 2 | 1

1 | 2 | 2

1 | 2 | 4

1 | 2 | 5

1 | 3 | 1

1 | 3 | 2

1 | 3 | 3

1 | 3 | 4

1 | 4 | 1

1 | 4 | 3

1 | 4 | 4

1 | 4 | 5

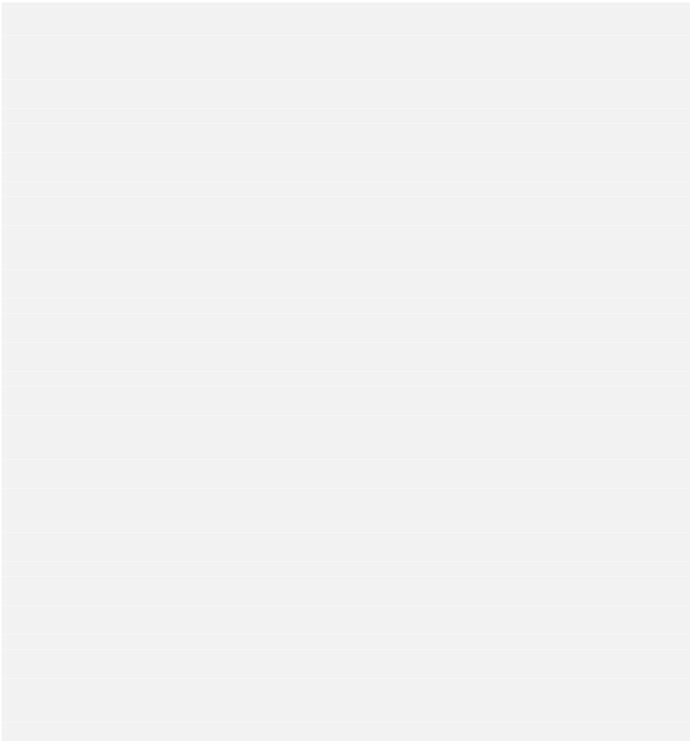
1 | 5 | 1

1 | 5 | 2

1 | 5 | 3

255 | 5 | 4

1 | 5 | 5



PostGIS 2.2.0dev 开发手册

436 / 686

**9.7**

**9.7.1**

**栅格编辑函数**

**ST\_SetGeoReference**

ST\_SetGeoReference —单次函数调用中设置6个地理参考值。参数之间相互用空格隔开。接受的输入类型可以是GDAL或ESRI格式，默认是GDAL格式

**用法**

raster ST\_SetGeoReference(raster rast, text georefcoords, text format=GDAL);

raster ST\_SetGeoReference(raster rast, double precision upperleftx, double precision upperlefty, double precision scalex, double

precision scaley, double precision skewx, double precision skewy);

**描述**

单次函数调用中设置6个地理参考值。接受的输入类型可以是GDAL或ESRI格式，默认是GDAL格式。如果没有提供6个值，该函数将会返回NULL

GDAL和ESRI格式如下

GDAL格式:

scalex skewy skewx scaley upperleftx upperlefty

ESRI格式:

scalex skewy skewx scaley upperleftx + scalex\*0.5 upperlefty + scaley\*0.5

**注意**

如果栅格有数据库之外的波段，修改地理参考可能会导致栅格外部存储数据的不正确访问

版本提升： 2.1.0 添加了函数形式ST\_SetGeoReference(raster, double precision, ...)

**样例**

WITH foo AS (

SELECT ST\_MakeEmptyRaster(5, 5, 0, 0, 1,

)

SELECT

0 AS rid, (ST\_Metadata(rast)).\*

FROM foo

UNION ALL

SELECT

1, (ST\_Metadata(ST\_SetGeoReference(rast,

FROM foo

UNION ALL

SELECT

2, (ST\_Metadata(ST\_SetGeoReference(rast,

FROM foo

UNION ALL

SELECT

3, (ST\_Metadata(ST\_SetGeoReference(rast,

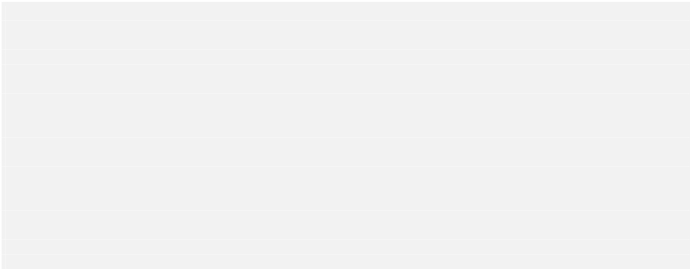
FROM foo

-1, 0, 0, 0) AS rast

’10 0 0 -10 0.1 0.1’, ’GDAL’))).\*

’10 0 0 -10 5.1 -4.9’, ’ESRI’))).\*

1, 1, 10, -10, 0.001, 0.001))).\*



PostGIS 2.2.0dev 开发手册

437 / 686

rid |

upperleftx

|

upperlefty

| width | height | scalex | scaley | skewx | ←

skewy | srid | numbands

-----+--------------------+--------------------+-------+--------+--------+--------+-------+-------+--

0 |

0 |

0 |

5 |

5 |

1 |

-1 |

0 | ←

0 |

0 |

0

1 |

0.1 | 0.1 |

5 |

5 |

10 |

-10 |

0 | ←

0 | 0 | 0

2 | 0.0999999999999996 | 0.0999999999999996 |

5 |

5 |

10 |

-10 |

0 | ←

0 | 0 | 0

3 | 1 | 1 |

5 |

5 |

10 |

-10 | 0.001 | ←

0.001 | 0 | 0

**相关请参考**

ST\_GeoReference, ST\_ScaleX, ST\_ScaleY, ST\_UpperLeftX, ST\_UpperLeftY

**9.7.2**

**ST\_SetRotation**

ST\_SetRotation —设置栅格的旋转弧度

**用法**

ﬂoat8 ST\_SetRotation(raster rast, ﬂoat8 rotation);

**描述**

统一设置栅格的旋转弧度。参考World File（https://en.wikipedia.org/wiki/World\_file）获取更多信息.

**样例**

SELECT

ST\_ScaleX(rast1), ST\_ScaleY(rast1), ST\_SkewX(rast1), ST\_SkewY(rast1),

ST\_ScaleX(rast2), ST\_ScaleY(rast2), ST\_SkewX(rast2), ST\_SkewY(rast2)

FROM (

SELECT ST\_SetRotation(rast, 15) AS rast1, rast as rast2 FROM dummy\_rast

) AS foo;

st\_scalex

|

st\_scaley

|

st\_skewx

|

st\_skewy

|

←

st\_scalex | st\_scaley | st\_skewx | st\_skewy

---------------------+---------------------+--------------------+--------------------+-----------+---

-1.51937582571764 | -2.27906373857646 | 1.95086352047135 | 1.30057568031423 |

2 | 3 | 0 | 0

-0.0379843956429411 | -0.0379843956429411 | 0.0325143920078558 | 0.0325143920078558 |

0.05 | -0.05 | 0 | 0

**相关请参考**

ST\_Rotation, ST\_ScaleX, ST\_ScaleY, ST\_SkewX, ST\_SkewY

←

←

**9.7.3**

**ST\_SetScale**

ST\_SetScale —设置像元的X和Y值，单位以空间参考系的坐标单位为准，就是设置像元的宽度和高度。

PostGIS 2.2.0dev 开发手册

438 / 686

**用法**

raster ST\_SetScale(raster rast, ﬂoat8 xy);

raster ST\_SetScale(raster rast, ﬂoat8 x, ﬂoat8 y);

**描述**

设置像元的X和Y值，单位以空间参考系的坐标单位为准，就是设置像元的宽度和高度。如果只传递了一个值，就假定X和Y值一样。

**注意**

ST\_SetScale 和函数ST\_Rescale不同的是函数ST\_SetScale不对栅格做重复采样来匹配栅格的边界。它只改变栅格的元数据信息或空间参考，以便纠正原有像元大小。而函数ST\_Rescale 生成一个为了适应输入栅格边界而改变宽度和高度大小的新栅格。ST\_SetScale函数不修改栅格的高度和宽度。

版本变更：2.0.0 ，在WKTRaster 版本中，该函数称为ST\_SetPixelSize，在2.0.0版本中更新成现在的名字

**样例**

UPDATE dummy\_rast

SET rast = ST\_SetScale(rast, 1.5)

WHERE rid = 2;

SELECT ST\_ScaleX(rast) As pixx, ST\_ScaleY(rast) As pixy, Box3D(rast) As newbox

FROM dummy\_rast

WHERE rid = 2;

pixx | pixy |

newbox

------+------+----------------------------------------------

1.5 | 1.5 | BOX(3427927.75 5793244 0, 3427935.25 5793251.5 0)

UPDATE dummy\_rast

SET rast = ST\_SetScale(rast, 1.5, 0.55)

WHERE rid = 2;

SELECT ST\_ScaleX(rast) As pixx, ST\_ScaleY(rast) As pixy, Box3D(rast) As newbox

FROM dummy\_rast

WHERE rid = 2;

pixx | pixy |

newbox

------+------+--------------------------------------------

1.5 | 0.55 | BOX(3427927.75 5793244 0,3427935.25 5793247 0)

**相关请参考**

ST\_ScaleX, ST\_ScaleY, Box3D

**9.7.4**

**ST\_SetSkew**

ST\_SetSkew — 设置栅格的X和Y倾斜度（或者说旋转参数）。如果只传递一个参数，那么会将X和Y值设置相同。



PostGIS 2.2.0dev 开发手册

439 / 686

**用法**

raster ST\_SetSkew(raster rast, ﬂoat8 skewxy);

raster ST\_SetSkew(raster rast, ﬂoat8 skewx, ﬂoat8 skewy);

**描述**

设置栅格的X和Y倾斜度（或者说旋转参数）。如果只传递一个参数，那么会将X和Y值设置相同。参考World File（https://en.wikipedia.org/wiki/World\_file）获取更多信息

**样例**

-- 样例 1

UPDATE dummy\_rast SET rast = ST\_SetSkew(rast,1,2) WHERE rid = 1;

SELECT rid, ST\_SkewX(rast) As skewx, ST\_SkewY(rast) As skewy,

ST\_GeoReference(rast) as georef

FROM dummy\_rast WHERE rid = 1;

rid | skewx | skewy |

georef

----+-------+-------+--------------

1 | 1 | 2 | 2.0000000000

: 2.0000000000

: 1.0000000000

: 3.0000000000

: 0.5000000000

: 0.5000000000

-- 样例 2 set both to same number:

UPDATE dummy\_rast SET rast = ST\_SetSkew(rast,0) WHERE rid = 1;

SELECT rid, ST\_SkewX(rast) As skewx, ST\_SkewY(rast) As skewy,

ST\_GeoReference(rast) as georef

FROM dummy\_rast WHERE rid = 1;

rid | skewx | skewy |

georef

-----+-------+-------+--------------

1 | 0 | 0 | 2.0000000000

: 0.0000000000

: 0.0000000000

: 3.0000000000

: 0.5000000000

: 0.5000000000

**相关请参考**

ST\_GeoReference, ST\_SetGeoReference, ST\_SkewX, ST\_SkewY

**9.7.5**

**ST\_SetSRID**

ST\_SetSRID — 设置栅格的SRID值

**用法**

raster ST\_SetSRID(raster rast, integer srid);

PostGIS 2.2.0dev 开发手册

440 / 686

**描述**

设置栅格的SRID值

**注意**

该函数不以任何方式转换栅格。它只是改变了栅格的参考系信息。对于后面的栅格空间转换很有用

**相关请参考**

Section 4.3.1, ST\_SRID

**9.7.6**

**ST\_SetUpperLeft**

ST\_SetUpperLeft — 把左上角的的像元的值设置为投影的X和Y坐标。

**用法**

raster ST\_SetUpperLeft(raster rast, double precision x, double precision y);

**描述**

把左上角的的像元的值设置为投影的X和Y坐标。

**样例**

SELECT ST\_SetUpperLeft(rast,-71.01,42.37)

FROM dummy\_rast

WHERE rid = 2;

**相关请参考**

ST\_UpperLeftX, ST\_UpperLeftY

**9.7.7**

**ST\_Resample**

ST\_Resample —使用一个指定的重采样算法、新维度参数（width和height）和任意一个网格角点和一个由其他栅格定义的栅格空间参考属性（由函数的形式参数ref指定）来重采样一个栅格。

**用法**

raster ST\_Resample(raster rast, integer width, integer height, double precision gridx=NULL, double precision gridy=NULL,

double precision skewx=0, double precision skewy=0, text algorithm=NearestNeighbour, double precision maxerr=0.125);

raster ST\_Resample(raster rast, double precision scalex=0, double precision scaley=0, double precision gridx=NULL, double

precision gridy=NULL, double precision skewx=0, double precision skewy=0, text algorithm=NearestNeighbor, double preci-

sion maxerr=0.125);

raster ST\_Resample(raster rast, raster ref, text algorithm=NearestNeighbour, double precision maxerr=0.125, boolean usescale=true);

raster ST\_Resample(raster rast, raster ref, boolean usescale, text algorithm=NearestNeighbour, double precision maxerr=0.125);



PostGIS 2.2.0dev 开发手册

441 / 686

**描述**

使用一个指定的重采样算法、新维度参数（width和height）和任意一个网格角点(由参数gridx和gridy指定)和一个栅格空间参考属性（(scalex, scaley, skewx & skewy）或引用其他栅格属性来重采样一个栅格。如果使用一个参考空间栅格，那么两个栅格必须拥有相同的SRID。

新的像素值会使用NearestNeighbor (English 或 American 单词拼写方法，译者注：英式和美式对同一个单词可能有不同的写法), Bilinear, Cubic, CubicSpline 或Lanczos 重采样算法来进行计算。默认的是NearestNeighbor算法，这个算法是最快的，但会产生最多的插值错误。如果没有指定maxerror，默认参数maxwhich值为0.125。

**注意**

参考 GDAL Warp resampling methods（http://www.gdal.org/gdalwarp.html）获取更多信息.

可用版本：2.0.0 需要 GDAL版本 1.6.1+

版本变更：2.1.0 参数SRID被移除了。带有引用参考栅格的函数变体不再使永引用栅格的SRID值，使用ST\_Transform() 来重投影栅格。支持无SRID的栅格.

**样例**

SELECT

ST\_Width(orig) AS orig\_width,

ST\_Width(reduce\_100) AS new\_width

FROM (

SELECT

rast AS orig,

ST\_Resample(rast,100,100) AS reduce\_100

FROM aerials.boston

WHERE ST\_Intersects(rast,

ST\_Transform(

ST\_MakeEnvelope(-71.128, 42.2392,-71.1277, 42.2397, 4326),26986)

)

LIMIT 1

) AS foo;

orig\_width | new\_width

------------+-------------

200 | 100

**相关请参考**

ST\_Rescale, ST\_Transform

**9.7.8**

**ST\_Rescale**

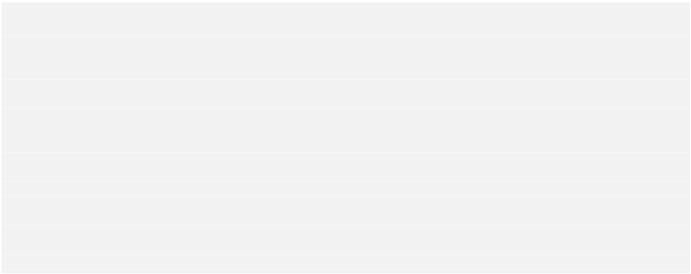
ST\_Rescale —重采样一个栅格，只调整像素的大小，新的像素值会使用NearestNeighbor (English 或 American 单词拼写方法，译者注：英式和美式对同一个单词可能有不同的写法), Bilinear, Cubic, CubicSpline 或Lanczos 重采样算法来进行计算。默认的是NearestNeighbor算法

**用法**

raster ST\_Rescale(raster rast, double precision scalexy, text algorithm=NearestNeighbour, double precision maxerr=0.125);

raster ST\_Rescale(raster rast, double precision scalex, double precision scaley, text algorithm=NearestNeighbour, double preci-

sion maxerr=0.125);



PostGIS 2.2.0dev 开发手册

442 / 686

**描述**

重采样一个栅格，只调整像素的大小，新的像素值会使用NearestNeighbor (English 或 American 单词拼写方法，译者注：英式和美式对同一个单词可能有不同的写法), Bilinear, Cubic, CubicSpline 或Lanczos 重采样算法来进行计算。默认的是NearestNeighbor算法，这个算法是最快的，但会产生最多的插值错误。

参数scalex 和参数 scaley定义一个新像素的size（大小）。scaley必须要减小以便适应原始栅格的边界。

如果像素新的参数scalex 和 scaley值不是栅格宽度和高度的因子（比如15的因子是1,3,5），那么函数返回的栅格的边界会扩展包围提供的栅格。

如果没有指定maxerror，默认参数maxwhich值为0.125。

**注意**

参考: GDAL Warp resampling methods获取更多信息

**注意**

函数ST\_Rescale与函数ST\_SetScale不同的地方是函数ST\_SetScale 不对栅格重采样以便适应原始栅格的边界。

函数ST\_SetScale值改变栅格的元数据信息(或者地理参考)纠正原有的缩放。

函数ST\_Rescale生成一个栅格，该栅格有不同的宽度和高度来适应输入栅格的边界。

函数ST\_SetScale不修改栅格的宽度和高度。

可用版本：2.0.0 需要版本 GDAL 1.6.1+

版本变更：2.1.0 支持没有SRID的栅格

**样例**

A simple example rescaling a raster from a pixel size of 0.001 degree to a pixel size of 0.0015 degree.

-- the original raster pixel size

SELECT ST\_PixelWidth(ST\_AddBand(ST\_MakeEmptyRaster(100, 100, 0, 0, 0.001, -0.001, 0, 0,

4269), ’8BUI’::text, 1, 0)) width

width

----------

0.001

-- the rescaled raster raster pixel size

←

SELECT ST\_PixelWidth(ST\_Rescale(ST\_AddBand(ST\_MakeEmptyRaster(100, 100, 0, 0, 0.001,

-0.001, 0, 0, 4269), ’8BUI’::text, 1, 0), 0.0015)) width

width

----------

0.0015

**相关请参考**

ST\_SetScale, ST\_ScaleX, ST\_ScaleY, ST\_Resample, ST\_Transform

←

**9.7.9**

**ST\_Reskew**

ST\_Reskew —重采样一个栅格，只调整像素的skew(或者说旋转参数)，新的像素值会使用NearestNeighbor (English 或 American 单词拼写方法，译者注：英式和美式对同一个单词可能有不同的写法), Bilinear, Cubic, CubicSpline 或Lanczos 重采样算法来进行计算。默认的是NearestNeighbor算法



PostGIS 2.2.0dev 开发手册

443 / 686

**用法**

raster ST\_Reskew(raster rast, double precision skewxy, text algorithm=NearestNeighbour, double precision maxerr=0.125);

raster ST\_Reskew(raster rast, double precision skewx, double precision skewy, text algorithm=NearestNeighbour, double preci-

sion maxerr=0.125);

**描述**

重采样一个栅格，只调整像素的skew(或者说旋转参数)，新的像素值会使用NearestNeighbor (English 或 American 单词拼写方法，译者注：英式和美式对同一个单词可能有不同的写法), Bilinear, Cubic, CubicSpline 或Lanczos 重采样算法来进行计算。默认的是NearestNeighbor算法，这个算法是最快的，但会产生最多的插值错误。

参数skewx和参数skewy定义了新栅格的skew(倾斜度)。

新栅格的边界会提供的栅格边界相遇（即相交）。

如果没有指定maxerror，默认参数maxwhich值为0.125。

**注意**

参考 GDAL Warp resampling methods（http://www.gdal.org/gdalwarp.html）获取更多信息.

**注意**

函数ST\_Reskew与函数ST\_SetSkew不同的地方是函数ST\_SetSkew不对栅格重采样以便适应原始栅格的边界。

函数ST\_SetSkew值改变栅格的元数据信息(或者地理参考)来纠正原有的倾斜度。

函数ST\_Reskew生成一个栅格，该栅格有不同的宽度和高度来适应输入栅格的边界。

函数ST\_SetSkew不修改栅格的宽度和高度。

可用版本：2.0.0 需要 GDAL 1.6.1+版本

版本变更：2.1.0 支持无SRID的栅格

**样例**

A simple example reskewing a raster from a skew of 0.0 to a skew of 0.0015.

-- the original raster pixel size

SELECT ST\_Rotation(ST\_AddBand(ST\_MakeEmptyRaster(100, 100, 0, 0, 0.001, -0.001, 0, 0, 4269) ←

, ’8BUI’::text, 1, 0))

-- the rescaled raster raster pixel size

SELECT ST\_Rotation(ST\_Reskew(ST\_AddBand(ST\_MakeEmptyRaster(100, 100, 0, 0, 0.001, -0.001,

0, 0, 4269), ’8BUI’::text, 1, 0), 0.0015))

**相关请参考**

ST\_Resample, ST\_Rescale, ST\_SetSkew, ST\_SetRotation, ST\_SkewX, ST\_SkewY, ST\_Transform

←

**9.7.10**

**ST\_SnapToGrid**

ST\_SnapToGrid —重采样一个栅格，方法是把栅格分割成由任意一个角点像素（gridx & gridy）和可选的像素大小（scalex和scaley）定义的网格。新的像素值会使用NearestNeighbor (English 或 American 单词拼写方法，译者注：英式和美式对同一个单词可能有不同的写法), Bilinear, Cubic, CubicSpline 或Lanczos 重采样算法来进行计算。默认的是NearestNeighbor算法



PostGIS 2.2.0dev 开发手册

444 / 686

**用法**

raster ST\_SnapToGrid(raster rast, double precision gridx, double precision gridy, text algorithm=NearestNeighbour, double

precision maxerr=0.125, double precision scalex=DEFAULT 0, double precision scaley=DEFAULT 0);

raster ST\_SnapToGrid(raster rast, double precision gridx, double precision gridy, double precision scalex, double precision

scaley, text algorithm=NearestNeighbour, double precision maxerr=0.125);

raster ST\_SnapToGrid(raster rast, double precision gridx, double precision gridy, double precision scalexy, text algorithm=NearestNeigh

double precision maxerr=0.125);

**描述**

重采样一个栅格，方法是把栅格分割成由任意一个角点像素（gridx & gridy）和可选的像素大小（scalex和scaley）定义的网格。新的像素值会使用NearestNeighbor (English 或 American 单词拼写方法，译者注：英式和美式对同一个单词可能有不同的写法), Bilinear, Cubic, CubicSpline 或Lanczos 重采样算法来进行计算。默认的是NearestNeighbor算法，这个算法是最快的，但会产生最多的插值错误。

参数gridx和参数gridy定义了新网格的任意角点。这对新栅格的左上角点不是必须的参数。角点也不必在新栅格边界里面或者边上。你可以选择性定义新网格的像素size（即通过参数scalex和scaley）。

新栅格的边界会提供的栅格边界相遇（即相交）。

如果没有指定maxerror，默认参数maxwhich值为0.125。

**注意**

参考 GDAL Warp resampling methods（http://www.gdal.org/gdalwarp.html）获取更多信息.

**注意**

如果你想控制更多关于栅格参数的使用，请使用函数ST\_Resample

可用版本：2.0.0 需要 GDAL版本 1.6.1+

版本变更：2.1.0 支持无SRID的栅格

**样例**

A simple example snapping a raster to a slightly different grid.

-- the original raster pixel size

SELECT ST\_UpperLeftX(ST\_AddBand(ST\_MakeEmptyRaster(10, 10, 0, 0, 0.001, -0.001, 0, 0, 4269) ←

, ’8BUI’::text, 1, 0))

-- the rescaled raster raster pixel size

SELECT ST\_UpperLeftX(ST\_SnapToGrid(ST\_AddBand(ST\_MakeEmptyRaster(10, 10, 0, 0, 0.001,

-0.001, 0, 0, 4269), ’8BUI’::text, 1, 0), 0.0002, 0.0002))

**相关请参考**

ST\_Resample, ST\_Rescale, ST\_UpperLeftX, ST\_UpperLeftY

←

**9.7.11**

**ST\_Resize**

ST\_Resize —重新设定栅格的高度和宽度



PostGIS 2.2.0dev 开发手册

445 / 686

**用法**

raster ST\_Resize(raster rast, integer width, integer height, text algorithm=NearestNeighbor, double precision maxerr=0.125);

raster ST\_Resize(raster rast, double precision percentwidth, double precision percentheight, text algorithm=NearestNeighbor,

double precision maxerr=0.125);

raster ST\_Resize(raster rast, text width, text height, text algorithm=NearestNeighbor, double precision maxerr=0.125);

**描述**

重新设定栅格的高度和宽度。新的宽度和高度可以设定为像元的个数或者原栅格的高度和宽度的百分比。

新的像素值会使用NearestNeighbor (English 或 American 单词拼写方法，译者注：英式和美式对同一个单词可能有不同的写法), Bilinear, Cubic, CubicSpline 或Lanczos 重采样算法来进行计算。默认的是NearestNeighbor算法，这个算法是最快的，但会产生最多的插值错误。

函数变体1需要输出栅格的实际参数width和参数height

函数变体2需要表示占有输入栅格的width和height比例的decimal值，范围在0到1之间。

函数变体3需要输出栅格的实际参数width和参数height值或者表示占有输入栅格的width和height比例的decimal值，范围在0到1之间。（译者注：即函数变体3是函数变体1和函数变体2的综合）。

可用版本：2.1.0 需要 GDAL版本 1.6.1+

**样例**

WITH foo AS(

SELECT

1 AS rid,

ST\_Resize(

ST\_AddBand(

ST\_MakeEmptyRaster(1000, 1000, 0, 0, 1,

, 1, ’8BUI’, 255, 0

)

, ’50%’, ’500’) AS rast

UNION ALL

SELECT

2 AS rid,

ST\_Resize(

ST\_AddBand(

ST\_MakeEmptyRaster(1000, 1000, 0, 0, 1,

, 1, ’8BUI’, 255, 0

)

, 500, 100) AS rast

UNION ALL

SELECT

3 AS rid,

ST\_Resize(

ST\_AddBand(

ST\_MakeEmptyRaster(1000, 1000, 0, 0, 1,

, 1, ’8BUI’, 255, 0

)

, 0.25, 0.9) AS rast

), bar AS (

SELECT rid, ST\_Metadata(rast) AS meta, rast

)

SELECT rid, (meta).\* FROM bar

-1, 0, 0, 0)

-1, 0, 0, 0)

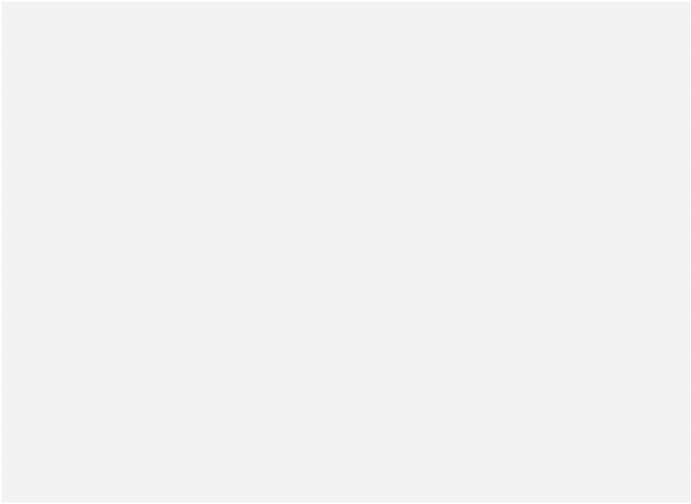
-1, 0, 0, 0)

FROM foo

rid | upperleftx | upperlefty | width | height | scalex | scaley | skewx | skewy | srid |

numbands

←



PostGIS 2.2.0dev 开发手册

446 / 686

-----+------------+------------+-------+--------+--------+--------+-------+-------+------+----------

1 |

0 |

0 |

500 |

500 |

1 |

-1 |

0 |

0 |

0 |

←

1

2 |

0 |

0 |

500 |

100 |

1 |

-1 |

0 |

0 |

0 |

←

1

3 |

0 |

0 |

250 |

900 |

1 |

-1 |

0 |

0 |

0 |

←

1

(3 rows)

**相关请参考**

ST\_Resample, ST\_Rescale, ST\_Reskew, ST\_SnapToGrid

**9.7.12**

**ST\_Transform**

ST\_Transform —根据给定的投影算法，将一个栅格的一种空间参考系转换到另一个空间参考系中。提供的投影算法有NearestNeighbor, Bilinear, Cubic, CubicSpline, Lanczos ，默认是NearestNeighbor算法

**用法**

raster ST\_Transform(raster rast, integer srid, text algorithm=NearestNeighbor, double precision maxerr=0.125, double precision

scalex, double precision scaley);

raster ST\_Transform(raster rast, integer srid, double precision scalex, double precision scaley, text algorithm=NearestNeighbor,

double precision maxerr=0.125);

raster ST\_Transform(raster rast, raster alignto, text algorithm=NearestNeighbor, double precision maxerr=0.125);

**描述**

通过指定的像素变形算法把一个栅格从一个已知的空间参考系变换到另一个已知的空间参考系。如果没有指定算法，那么使用算法’NearestNeighbor’，如果没有指定参数maxerr，那么默认使用参数0.125。可选的算法有：’NearestNeighbor’, ’Bilinear’, ’Cubic’, ’CubicSpline’, and ’Lanczos’。参考： [GDAL Warp resamplingmethods](http://www.gdal.org/gdalwarp.html)获取更多细节。

函数ST\_Transform进程会和函数ST\_SetSRID()弄混淆。实际上函数ST\_Transform只把栅格的坐标从一个参考系转换成另一个参考系下的坐标(同时也重采样栅格的值)，而函数ST\_SetSRID()值修改栅格的SRID值。

与其他函数变体不同，函数变体3需要一个用于对齐的引用栅格即参数alignto表示的栅格。被转换的栅格将会被转换到引用栅格的所在的空间参考系下（SRID值一致），并且对齐方式也和引用栅格的对齐方式一致。 (通过参数ST\_SameAlignment = TRUE控制)

**注意**

如果你发现没有正确地转换，你可能需要设置你的PostGIS的环境变量PROJSO所对应的投影库文件路径，对Linux来说是.so，对Windows来说是.dll文件。比如对于Windows，设置方法如下Control Panel（控制面板） -> System -> Environment Variables，添加一个叫做PROJSO的环境变量，并且设置值为libproj.dll (如果你使用库版本是proj 4.6.1)。修改这个参数后，你需要重启PostgreSQL服务

可用版本：2.0.0 需要 GDAL版本 1.6.1+

版本提升： 2.1.0 添加函数变体支持ST\_Transform(rast, alignto)



PostGIS 2.2.0dev 开发手册

447 / 686

**样例**

SELECT ST\_Width(mass\_stm) As w\_before, ST\_Width(wgs\_84) As w\_after,

ST\_Height(mass\_stm) As h\_before, ST\_Height(wgs\_84) As h\_after

FROM

( SELECT rast As mass\_stm, ST\_Transform(rast,4326) As wgs\_84

,

ST\_Transform(rast,4326, ’Bilinear’) AS wgs\_84\_bilin

FROM aerials.o\_2\_boston

WHERE ST\_Intersects(rast,

ST\_Transform(ST\_MakeEnvelope(-71.128, 42.2392,-71.1277, 42.2397, 4326),26986) )

LIMIT 1) As foo;

w\_before | w\_after | h\_before | h\_after

----------+---------+----------+---------

200 | 228 | 200 | 170

original mass state plane meters

After transform to wgs 84 long lat

(wgs\_84)

After transform to wgs 84 long lat

with bilinear algorithm instead of NN

default (wgs\_84\_bilin)

(mass\_stm)

**样例: Variant 3**

The following shows the difference between using ST\_Transform(raster, srid) and ST\_Transform(raster, alignto)

WITH foo AS (

SELECT 0 AS rid, ST\_AddBand(ST\_MakeEmptyRaster(2, 2, -500000, 600000, 100, -100, 0, 0,

←

2163), 1, ’16BUI’, 1, 0) AS rast UNION ALL

SELECT 1, ST\_AddBand(ST\_MakeEmptyRaster(2, 2, -499800, 600000, 100, -100, 0, 0, 2163), 1, ←

’16BUI’, 2, 0) AS rast UNION ALL

SELECT 2, ST\_AddBand(ST\_MakeEmptyRaster(2, 2, -499600, 600000, 100, -100, 0, 0, 2163), 1, ←

’16BUI’, 3, 0) AS rast UNION ALL

SELECT 3, ST\_AddBand(ST\_MakeEmptyRaster(2, 2, -500000, 599800, 100, -100, 0, 0, 2163), 1, ←

’16BUI’, 10, 0) AS rast UNION ALL

SELECT 4, ST\_AddBand(ST\_MakeEmptyRaster(2, 2, -499800, 599800, 100, -100, 0, 0, 2163), 1, ←

’16BUI’, 20, 0) AS rast UNION ALL

SELECT 5, ST\_AddBand(ST\_MakeEmptyRaster(2, 2, -499600, 599800, 100, -100, 0, 0, 2163), 1, ←

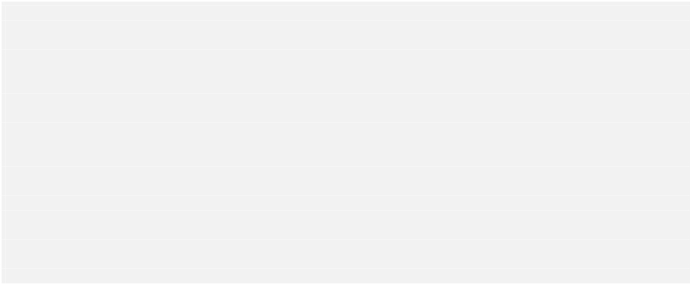
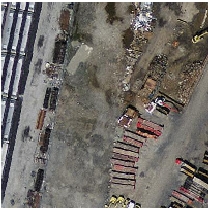
’16BUI’, 30, 0) AS rast UNION ALL

SELECT 6, ST\_AddBand(ST\_MakeEmptyRaster(2, 2, -500000, 599600, 100, -100, 0, 0, 2163), 1, ←

’16BUI’, 100, 0) AS rast UNION ALL

SELECT 7, ST\_AddBand(ST\_MakeEmptyRaster(2, 2, -499800, 599600, 100, -100, 0, 0, 2163), 1, ←

’16BUI’, 200, 0) AS rast UNION ALL



PostGIS 2.2.0dev 开发手册

448 / 686

SELECT 8, ST\_AddBand(ST\_MakeEmptyRaster(2, 2, -499600, 599600, 100, -100, 0, 0, 2163), 1, ←

’16BUI’, 300, 0) AS rast

), bar AS (

SELECT

ST\_Transform(rast, 4269) AS alignto

FROM foo

LIMIT 1

), baz AS (

SELECT

rid,

rast,

ST\_Transform(rast, 4269) AS not\_aligned,

ST\_Transform(rast, alignto) AS aligned

FROM foo

CROSS JOIN bar

)

SELECT

ST\_SameAlignment(rast) AS rast,

ST\_SameAlignment(not\_aligned) AS not\_aligned,

ST\_SameAlignment(aligned) AS aligned

FROM baz

rast | not\_aligned | aligned

------+-------------+---------

t

| f

| t

**相关请参考**

ST\_Transform, ST\_SetSRID

**9.8**

**9.8.1**

**栅格波段编辑函数**

**ST\_SetBandNoDataValue**

ST\_SetBandNoDataValue —把给定的波段设置为NODATA值，如果没有指定波段，则默认波段1。要标记一个波段没有NODATA值，把NODTA值设置为NULL

**用法**

raster ST\_SetBandNoDataValue(raster rast, double precision nodatavalue);

raster ST\_SetBandNoDataValue(raster rast, integer band, double precision nodatavalue, boolean forcechecking=false);

**描述**

把给定的波段设置为NODATA值，如果没有指定波段，则默认波段1。这会影响函数ST\_Polygon,ST\_DumpAsPolygons, 和ST\_PixelAs...()这一类函数的结果

**样例**

-- change just first band no data value

UPDATE dummy\_rast

SET rast = ST\_SetBandNoDataValue(rast,1, 254)

WHERE rid = 2;



PostGIS 2.2.0dev 开发手册

449 / 686

-- change no data band value of bands 1,2,3

UPDATE dummy\_rast

SET rast =

ST\_SetBandNoDataValue(

ST\_SetBandNoDataValue(

ST\_SetBandNoDataValue(

rast,1, 254)

,2,99),

3,108)

WHERE rid = 2;

-- wipe out the nodata value this will ensure all pixels are considered for all processing

functions

UPDATE dummy\_rast

SET rast = ST\_SetBandNoDataValue(rast,1, NULL)

WHERE rid = 2;

**相关请参考**

ST\_BandNoDataValue, ST\_NumBands

←

**9.8.2**

**ST\_SetBandIsNoData**

ST\_SetBandIsNoData —把波段的isnodata标识未TRUE

**用法**

raster ST\_SetBandIsNoData(raster rast, integer band=1);

**描述**

把波段的isnodata标识未TRUE。如果没有指定波段，则默认为波段1。该函数只有在flag被认为损坏的情况下才应该调用。flag被损坏的意思是调用函数 ST\_BandIsNoData的最后一个参数值TRUE的结果和不传递该参数值的结果不同，则认为flag被损坏了

可用版本：2.0.0

**样例**

-- Create dummy table with one raster column

create table dummy\_rast (rid integer, rast raster);

-- Add raster with two bands, one pixel/band. In the first band, nodatavalue = pixel value

= 3.

-- In the second band, nodatavalue = 13, pixel value = 4

insert into dummy\_rast values(1,

(

’01’ -- little endian (uint8 ndr)

||

’0000’ -- version (uint16 0)

||

’0200’ -- nBands (uint16 0)

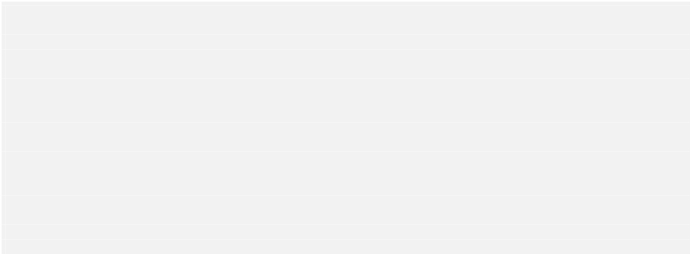
||

’17263529ED684A3F’ -- scaleX (float64 0.000805965234044584)

||

’F9253529ED684ABF’ -- scaleY (float64 -0.00080596523404458)

←



PostGIS 2.2.0dev 开发手册

450 / 686

||

’1C9F33CE69E352C0’ -- ipX (float64 -75.5533328537098)

||

’718F0E9A27A44840’ -- ipY (float64 49.2824585505576)

||

’ED50EB853EC32B3F’ -- skewX (float64 0.000211812383858707)

||

’7550EB853EC32B3F’ -- skewY (float64 0.000211812383858704)

||

’E6100000’ -- SRID (int32 4326)

||

’0100’ -- width (uint16 1)

||

’0100’ -- height (uint16 1)

||

’4’ -- hasnodatavalue set to true, isnodata value set to false (when it should be true)

||

’2’ -- first band type (4BUI)

||

’03’ -- novalue==3

||

’03’ -- pixel(0,0)==3 (same that nodata)

||

’0’ -- hasnodatavalue set to false

||

’5’ -- second band type (16BSI)

||

’0D00’ -- novalue==13

||

’0400’ -- pixel(0,0)==4

)::raster

);

select st\_bandisnodata(rast, 1) from dummy\_rast where rid = 1; -- Expected false

select st\_bandisnodata(rast, 1, TRUE) from dummy\_rast where rid = 1; -- Expected true

-- The isnodata flag is dirty. We are going to set it to true

update dummy\_rast set rast = st\_setbandisnodata(rast, 1) where rid = 1;

select st\_bandisnodata(rast, 1) from dummy\_rast where rid = 1; -- Expected true

**相关请参考**

ST\_BandNoDataValue, ST\_NumBands, ST\_SetBandNoDataValue, ST\_BandIsNoData

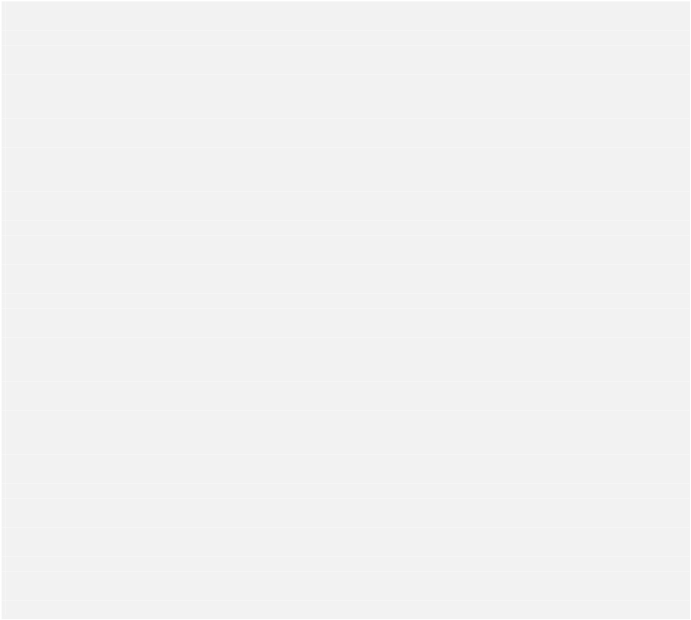
**9.9**

**9.9.1**

**栅格波段统计和分析函数**

**ST\_Count**

ST\_Count — 返回一个栅格给定波段的像素个数。如果没有指定波段，默认取波段1。如果参数exclude\_nodata\_value设置为true，那么这个函数只会统计像素值不为NODATA的像素个数。



PostGIS 2.2.0dev 开发手册

451 / 686

**用法**

bigint ST\_Count(raster rast, integer nband=1, boolean exclude\_nodata\_value=true);

bigint ST\_Count(raster rast, boolean exclude\_nodata\_value);

bigint ST\_Count(text rastertable, text rastercolumn, integer nband=1, boolean exclude\_nodata\_value=true);

bigint ST\_Count(text rastertable, text rastercolumn, boolean exclude\_nodata\_value);

**描述**

返回一个栅格给定波段的像素个数。如果没有指定波段，默认取波段1。

**注意**

如果参数exclude\_nodata\_value设置为true，那么这个函数只会统计像素值不为NODATA的像素个数。如果exclude\_nodata\_value值为false，那么将会统计所有的像素个数。

可用版本：2.0.0

**警告**

函数变体ST\_Count(rastertable, rastercolumn, ...) 在2.2.0版本中不推荐使用了，使用函数ST\_CountAgg来代替。

**样例**

--example will count all pixels not 249 and one will count all pixels.

--

SELECT rid, ST\_Count(ST\_SetBandNoDataValue(rast,249)) As exclude\_nodata,

ST\_Count(ST\_SetBandNoDataValue(rast,249),false) As include\_nodata

FROM dummy\_rast WHERE rid=2;

rid | exclude\_nodata | include\_nodata

-----+----------------+----------------

2 | 23 | 25

**相关请参考**

ST\_CountAgg, ST\_SummaryStats, ST\_SetBandNoDataValue

**9.9.2**

**ST\_CountAgg**

ST\_CountAgg — 聚合函数。返回一个栅格给定波段的像素个数。如果没有指定波段，默认取波段1。如果参数exclude\_nodata\_value设置为true，那么这个函数只会统计像素值不为NODATA的像素个数。

**用法**

bigint ST\_CountAgg(raster rast, integer nband, boolean exclude\_nodata\_value, double precision sample\_percent);

bigint ST\_CountAgg(raster rast, integer nband, boolean exclude\_nodata\_value);

bigint ST\_CountAgg(raster rast, boolean exclude\_nodata\_value);



PostGIS 2.2.0dev 开发手册

452 / 686

**描述**

聚合函数。返回一个栅格给定波段的像素个数。如果没有指定波段，默认取波段1。如果参数exclude\_nodata\_value设置为true，那么这个函数只会统计像素值不为NODATA的像素个数。如果exclude\_nodata\_value值为false，那么将会统计所有的像素个数。同时会默认重采样所有的像素，想要更快获得响应，把参数sample\_percent的值设置在0到1范围内。

可用版本：2.2.0

**样例**

WITH foo AS (

SELECT

rast.rast

FROM (

SELECT ST\_SetValue(

ST\_SetValue(

ST\_SetValue(

ST\_AddBand(

ST\_MakeEmptyRaster(10, 10, 10, 10, 2, 2, 0, 0,0)

, 1, ’64BF’, 0, 0

)

, 1, 1, 1, -10

)

, 1, 5, 4, 0

)

, 1, 5, 5, 3.14159

) AS rast

) AS rast

FULL JOIN (

SELECT generate\_series(1, 10) AS id

) AS id

ON 1 = 1

)

SELECT

ST\_CountAgg(rast, 1, TRUE)

FROM foo;

st\_countagg

-------------

20

(1 row)

**相关请参考**

ST\_Count, ST\_SummaryStats, ST\_SetBandNoDataValue

**9.9.3**

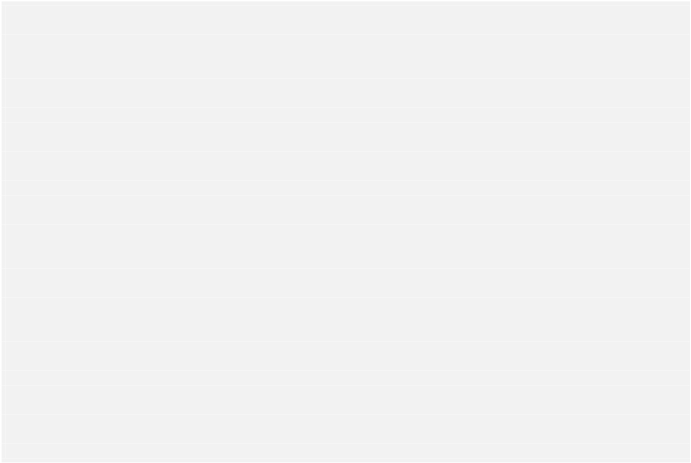
**ST\_Histogram**

ST\_Histogram —返回概括一个栅格或栅格覆盖数据分布的记录集。如果没有指定参数bins值，默认取autocomputed

**用法**

SETOF record ST\_Histogram(raster rast, integer nband=1, boolean exclude\_nodata\_value=true, integer bins=autocomputed,

double precision[] width=NULL, boolean right=false);



PostGIS 2.2.0dev 开发手册

453 / 686

SETOF record ST\_Histogram(raster rast, integer nband, integer bins, double precision[] width=NULL, boolean right=false);

SETOF record ST\_Histogram(raster rast, integer nband, boolean exclude\_nodata\_value, integer bins, boolean right);

SETOF record ST\_Histogram(raster rast, integer nband, integer bins, boolean right);

SETOF record ST\_Histogram(text rastertable, text rastercolumn, integer nband, integer bins, boolean right);

SETOF record ST\_Histogram(text rastertable, text rastercolumn, integer nband, boolean exclude\_nodata\_value, integer bins,

boolean right);

SETOF record ST\_Histogram(text rastertable, text rastercolumn, integer nband=1, boolean exclude\_nodata\_value=true, integer

bins=autocomputed, double precision[] width=NULL, boolean right=false);

SETOF record ST\_Histogram(text rastertable, text rastercolumn, integer nband=1, integer bins, double precision[] width=NULL,

boolean right=false);

**描述**

返回概括一个栅格波段数据分布情况：比如像素最小值，像素最大值，像素总数，像素值占比等统计值。如果没有指定波段，nband默认取1。

**注意**

如果参数exclude\_nodata\_value设置为true，那么这个函数只会统计像素值不为NODATA的像素个数。如果exclude\_nodata\_value值为false，那么将会统计所有的像素个数。

**width** double precision[] width：一个表示每一个直方图宽度的数组。如果分隔点个数比宽度数组个数多，那么宽度数组要进行重复赋值。样例: bins=9, widths are [a, b, c] 将会输出成 [a, b, c, a, b, c, a, b, c]

**bins**integer：设置柱状图分隔数——这个值表示函数返回的记录数。如果你没有指定，那么函数会取autocomputed，表示自动计算出。

**right**boolean：从右边而不是左边开始计算柱状图（默认）。也就是说一个区间右区间是闭区间。比如是 (a, b]而不是[a, b) 。

可用版本：2.0.0

**样例: Single raster tile - compute histograms for bands 1, 2, 3 and autocompute bins**

SELECT band, (stats).\*

FROM (SELECT rid, band, ST\_Histogram(rast, band) As stats

FROM dummy\_rast CROSS JOIN generate\_series(1,3) As band

WHERE rid=2) As foo;

band |

min

|

max

| count | percent

------+-------+-------+-------+---------

1 | 249 | 250 | 2 | 0.08

1 | 250 | 251 | 2 | 0.08

1 | 251 | 252 | 1 | 0.04

1 | 252 | 253 | 2 | 0.08

1 | 253 | 254 | 18 | 0.72

2 | 78 | 113.2 | 11 | 0.44

2 | 113.2 | 148.4 | 4 | 0.16

2 | 148.4 | 183.6 | 4 | 0.16

2 | 183.6 | 218.8 | 1 | 0.04

2 | 218.8 | 254 | 5 | 0.2

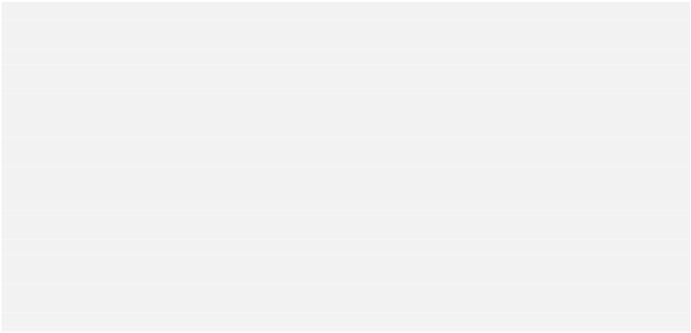
3 | 62 | 100.4 | 11 | 0.44

3 | 100.4 | 138.8 | 5 | 0.2

3 | 138.8 | 177.2 | 4 | 0.16

3 | 177.2 | 215.6 | 1 | 0.04

3 | 215.6 | 254 | 4 | 0.16



PostGIS 2.2.0dev 开发手册

454 / 686

**样例: Just band 2 but for 6 bins**

SELECT (stats).\*

FROM (SELECT rid, ST\_Histogram(rast, 2,6) As stats

FROM dummy\_rast

WHERE rid=2) As foo;

min

|

max

| count | percent

------------+------------+-------+---------

78 | 107.333333 | 9 | 0.36

107.333333 | 136.666667 | 6 | 0.24

136.666667 | 166 | 0 | 0

166 | 195.333333 | 4 | 0.16

195.333333 | 224.666667 | 1 | 0.04

224.666667 | 254 | 5 | 0.2

(6 rows)

-- Same as previous but we explicitly control the pixel value range of each bin.

SELECT (stats).\*

FROM (SELECT rid, ST\_Histogram(rast, 2,6,ARRAY[0.5,1,4,100,5]) As stats

FROM dummy\_rast

WHERE rid=2) As foo;

min

|

max

| count | percent

-------+-------+-------+----------

78 | 78.5 | 1 | 0.08

78.5 | 79.5 | 1 | 0.04

79.5 | 83.5 | 0 | 0

83.5 | 183.5 | 17 | 0.0068

183.5 | 188.5 | 0 | 0

188.5 | 254 | 6 | 0.003664

(6 rows)

**相关请参考**

ST\_Count, ST\_SummaryStats, ST\_SummaryStatsAgg

**9.9.4**

**ST\_Quantile**

ST\_Quantile — 计算一个栅格或者栅格覆盖的分位点。因此一个像素值可能是一个栅格的25%, 50%, 75% 的分位点。

**用法**

SETOF record ST\_Quantile(raster rast, integer nband=1, boolean exclude\_nodata\_value=true, double precision[] quantiles=NULL);

SETOF record ST\_Quantile(raster rast, double precision[] quantiles);

SETOF record ST\_Quantile(raster rast, integer nband, double precision[] quantiles);

double precision ST\_Quantile(raster rast, double precision quantile);

double precision ST\_Quantile(raster rast, boolean exclude\_nodata\_value, double precision quantile=NULL);

double precision ST\_Quantile(raster rast, integer nband, double precision quantile);

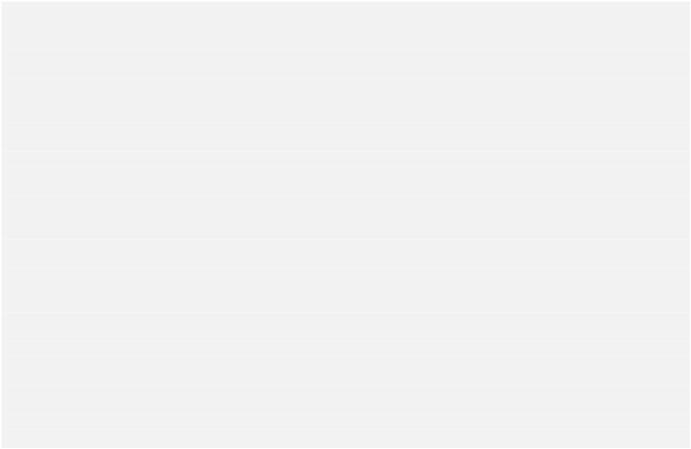
double precision ST\_Quantile(raster rast, integer nband, boolean exclude\_nodata\_value, double precision quantile);

double precision ST\_Quantile(raster rast, integer nband, double precision quantile);

SETOF record ST\_Quantile(text rastertable, text rastercolumn, integer nband=1, boolean exclude\_nodata\_value=true, double

precision[] quantiles=NULL);

SETOF record ST\_Quantile(text rastertable, text rastercolumn, integer nband, double precision[] quantiles);



PostGIS 2.2.0dev 开发手册

455 / 686

**描述**

计算一个栅格或者栅格覆盖的分位点。因此一个像素值可能是一个栅格的25%, 50%, 75% 的分位点。

**注意**

如果exclude\_nodata\_value值为false，那么将会统计所有的像素个数。

可用版本：2.0.0

**样例**

UPDATE dummy\_rast SET rast = ST\_SetBandNoDataValue(rast,249) WHERE rid=2;

--样例 will consider only pixels of band 1 that are not 249 and in named quantiles --

SELECT (pvq).\*

FROM (SELECT ST\_Quantile(rast, ARRAY[0.25,0.75]) As pvq

FROM dummy\_rast WHERE rid=2) As foo

ORDER BY (pvq).quantile;

quantile | value

----------+-------

0.25 | 253

0.75 | 254

SELECT ST\_Quantile(rast, 0.75) As value

FROM dummy\_rast WHERE rid=2;

value

------

254

--real live example.

Quantile of all pixels in band 2 intersecting a geometry

SELECT rid, (ST\_Quantile(rast,2)).\* As pvc

FROM o\_4\_boston

WHERE ST\_Intersects(rast,

ST\_GeomFromText(’POLYGON((224486 892151,224486 892200,224706 892200,224706

892151,224486 892151))’,26986)

)

ORDER BY value, quantile,rid

;

rid | quantile | value

-----+----------+-------

1 | 0 | 0

2 | 0 | 0

14 | 0 | 1

15 | 0 | 2

14 | 0.25 | 37

1 | 0.25 | 42

15 | 0.25 | 47

2 | 0.25 | 50

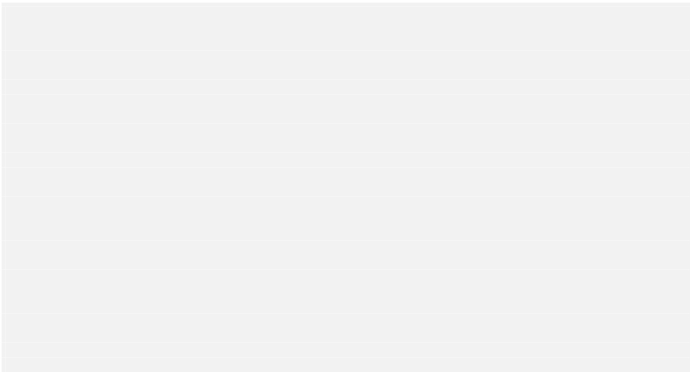
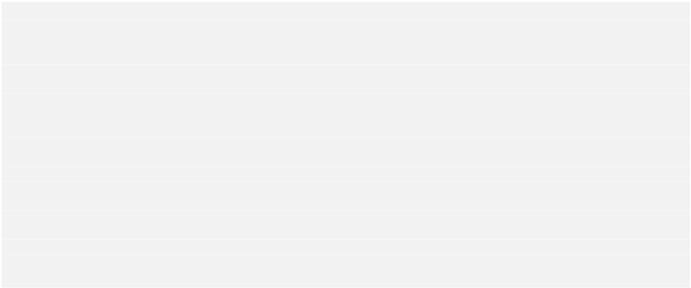
14 | 0.5 | 56

1 | 0.5 | 64

15 | 0.5 | 66

2 | 0.5 | 77

←



PostGIS 2.2.0dev 开发手册

456 / 686

14

15

1

2

14

1

2

15

|

|

|

|

|

|

|

|

0.75

0.75

0.75

0.75

1

1

1

1

|

|

|

|

|

|

|

|

81

87

94

106

199

244

255

255

**相关请参考**

ST\_Count, ST\_SummaryStats, ST\_SummaryStatsAgg, ST\_SetBandNoDataValue

**9.9.5**

**ST\_SummaryStats**

ST\_SummaryStats —返回一个栅格或者栅格覆盖的一个指定波段的概览统计值，包括：count（像素总数）, sum（像素值之和）, mean（像素平均值）, stddev（像素值标准差）, min（最小像素值）, max（最大像素值）。如果没有指定波段，默认取波段1。

**用法**

summarystats ST\_SummaryStats(raster rast, boolean exclude\_nodata\_value);

summarystats ST\_SummaryStats(raster rast, integer nband, boolean exclude\_nodata\_value);

summarystats ST\_SummaryStats(text rastertable, text rastercolumn, boolean exclude\_nodata\_value);

summarystats ST\_SummaryStats(text rastertable, text rastercolumn, integer nband=1, boolean exclude\_nodata\_value=true);

**描述**

返回一个栅格或者栅格覆盖的一个指定波段的概览统计值，包括：count（像素总数）, sum（像素值之和）, mean（像素平均值）, stddev（像素值标准差）, min（最小像素值）, max（最大像素值）。如果没有指定波段，默认取波段1。

**注意**

如果参数exclude\_nodata\_value设置为true，那么这个函数只会统计像素值不为NODATA的像素个数。如果exclude\_nodata\_value值为false，那么将会统计所有的像素个数。

**注意**

默认会采样所有的像素，要想更快获得结果，把参数sample\_percent值设置小于1。

可用版本：2.0.0

**警告**

函数变体ST\_SummaryStats(rastertable, rastercolumn, ...) 在PostGIS 2.2.0版本已经不推荐使用了，使用函数ST\_SummaryStatsAgg代替。



PostGIS 2.2.0dev 开发手册

457 / 686

**样例: Single raster tile**

SELECT rid, band, (stats).\*

FROM (SELECT rid, band, ST\_SummaryStats(rast, band) As stats

FROM dummy\_rast CROSS JOIN generate\_series(1,3) As band

WHERE rid=2) As foo;

rid | band | count | sum

|

mean

|

stddev

| min | max

-----+------+-------+------+------------+-----------+-----+-----

2 | 1 | 23 | 5821 | 253.086957 | 1.248061 | 250 | 254

2 | 2 | 25 | 3682 | 147.28 | 59.862188 | 78 | 254

2 | 3 | 25 | 3290 | 131.6 | 61.647384 | 62 | 254

**样例: Summarize pixels that intersect buildings of interest**

This example took 574ms on PostGIS windows 64-bit with all of Boston Buildings and aerial Tiles (tiles each 150x150 pixels ~

134,000 tiles), ~102,000 building records

WITH

-- our features of interest

feat AS (SELECT gid As building\_id, geom\_26986 As geom FROM buildings AS b

WHERE gid IN(100, 103,150)

),

-- clip band 2 of raster tiles to boundaries of builds

-- then get stats for these clipped regions

b\_stats AS

(SELECT

building\_id, (stats).\*

FROM (SELECT building\_id, ST\_SummaryStats(ST\_Clip(rast,2,geom)) As stats

FROM aerials.boston

INNER JOIN feat

ON ST\_Intersects(feat.geom,rast)

) As foo

)

-- finally summarize stats

SELECT building\_id, SUM(count) As num\_pixels

, MIN(min) As min\_pval

,

MAX(max) As max\_pval

, SUM(mean\*count)/SUM(count) As avg\_pval

FROM b\_stats

WHERE count > 0

GROUP BY building\_id

ORDER BY building\_id;

building\_id | num\_pixels | min\_pval | max\_pval |

avg\_pval

-------------+------------+----------+----------+------------------

100 | 1090 | 1 | 255 | 61.0697247706422

103 | 655 | 7 | 182 | 70.5038167938931

150 | 895 | 2 | 252 | 185.642458100559

**样例: Raster coverage**

-- stats for each band --

SELECT band, (stats).\*

FROM (SELECT band, ST\_SummaryStats(’o\_4\_boston’,’rast’, band) As stats

FROM generate\_series(1,3) As band) As foo;

band |

count

|

sum

|

mean

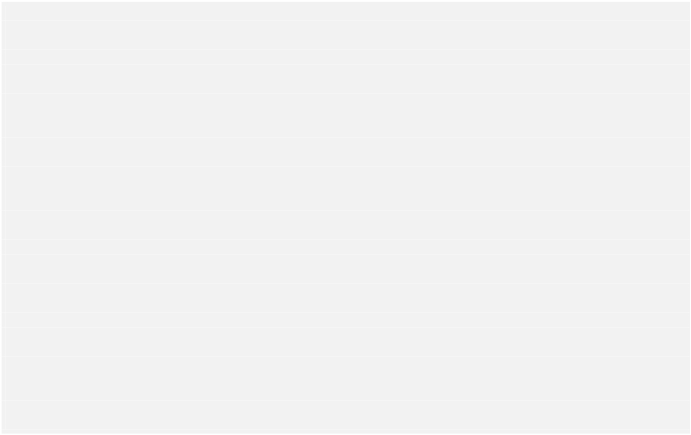
|

stddev

| min | max

------+---------+--------+------------------+------------------+-----+-----

1 | 8450000 | 725799 | 82.7064349112426 | 45.6800222638537 | 0 | 255



PostGIS 2.2.0dev 开发手册

458 / 686

2 | 8450000 | 700487 | 81.4197705325444 | 44.2161184161765 |

3 | 8450000 | 575943 | 74.682739408284 | 44.2143885481407 |

0 | 255

0 | 255

-- For a table -- will get better speed if set sampling to less than 100%

-- Here we set to 25% and get a much faster answer

SELECT band, (stats).\*

FROM (SELECT band, ST\_SummaryStats(’o\_4\_boston’,’rast’, band,true,0.25) As stats

FROM generate\_series(1,3) As band) As foo;

band |

count

|

sum

|

mean

|

stddev

| min | max

------+---------+--------+------------------+------------------+-----+-----

1 | 2112500 | 180686 | 82.6890480473373 | 45.6961043857248 | 0 | 255

2 | 2112500 | 174571 | 81.448503668639 | 44.2252623171821 | 0 | 255

3 | 2112500 | 144364 | 74.6765884023669 | 44.2014869384578 | 0 | 255

**相关请参考**

summarystats, ST\_SummaryStatsAgg, ST\_Count, ST\_Clip

**9.9.6**

**ST\_SummaryStatsAgg**

ST\_SummaryStatsAgg —聚合函数。返回一个栅格或者一个栅格集合的某个指定波段的概览统计值，包括：count（像素总数）, sum（像素值之和）, mean（像素平均值）, stddev（像素值标准差）, min（最小像素值）, max（最大像素值）。如果没有指定波段，默认取波段1。

**用法**

summarystats ST\_SummaryStatsAgg(raster rast, integer nband, boolean exclude\_nodata\_value, double precision sample\_percent);

summarystats ST\_SummaryStatsAgg(raster rast, boolean exclude\_nodata\_value, double precision sample\_percent);

summarystats ST\_SummaryStatsAgg(raster rast, integer nband, boolean exclude\_nodata\_value);

**描述**

聚合函数。返回一个栅格或者一个栅格集合的某个指定波段的概览统计值，包括：count（像素总数）, sum（像素值之和）, mean（像素平均值）, stddev（像素值标准差）, min（最小像素值）, max（最大像素值）。如果没有指定波段，默认取波段1。

**注意**

如果参数exclude\_nodata\_value设置为true，那么这个函数只会统计像素值不为NODATA的像素个数。如果exclude\_nodata\_value值为false，那么将会统计所有的像素个数。

**注意**

默认会采样所有的像素，要想更快获得结果，把参数sample\_percent值设置小于1。

可用版本：2.2.0

**样例**



PostGIS 2.2.0dev 开发手册

459 / 686

WITH foo AS (

SELECT

rast.rast

FROM (

SELECT ST\_SetValue(

ST\_SetValue(

ST\_SetValue(

ST\_AddBand(

ST\_MakeEmptyRaster(10, 10, 10, 10, 2, 2, 0, 0,0)

, 1, ’64BF’, 0, 0

)

, 1, 1, 1, -10

)

, 1, 5, 4, 0

)

, 1, 5, 5, 3.14159

) AS rast

) AS rast

FULL JOIN (

SELECT generate\_series(1, 10) AS id

) AS id

ON 1 = 1

)

SELECT

(stats).count,

round((stats).sum::numeric, 3),

round((stats).mean::numeric, 3),

round((stats).stddev::numeric, 3),

round((stats).min::numeric, 3),

round((stats).max::numeric, 3)

FROM (

SELECT

ST\_SummaryStatsAgg(rast, 1, TRUE, 1) AS stats

FROM foo

) bar;

count |

round

| round

| round |

round

| round

-------+---------+--------+-------+---------+-------

20 | -68.584 | -3.429 | 6.571 | -10.000 | 3.142

(1 row)

**相关请参考**

summarystats, ST\_SummaryStats, ST\_Count, ST\_Clip

**9.9.7**

**ST\_ValueCount**

ST\_ValueCount — 返回一个记录集，包括像素值和指定栅格（或栅格覆盖）的指定波段的像素值在一个值集合内的像素个数。如果没有指定波段，那么默认是波段1。默认也不统计值为NODATA的像素。像素值如果不是整数，那么像素值会进行round四舍五入处理得到一个最接近的整数值。

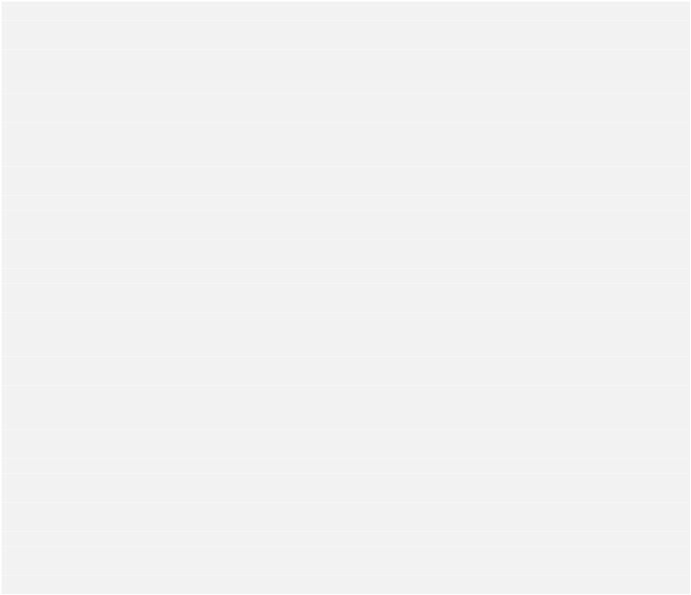
**用法**

SETOF record ST\_ValueCount(raster rast, integer nband=1, boolean exclude\_nodata\_value=true, double precision[] searchval-

ues=NULL, double precision roundto=0, double precision OUT value, integer OUT count);

SETOF record ST\_ValueCount(raster rast, integer nband, double precision[] searchvalues, double precision roundto=0, double

precision OUT value, integer OUT count);



PostGIS 2.2.0dev 开发手册

460 / 686

SETOF record ST\_ValueCount(raster rast, double precision[] searchvalues, double precision roundto=0, double precision OUT

value, integer OUT count);

bigint ST\_ValueCount(raster rast, double precision searchvalue, double precision roundto=0);

bigint ST\_ValueCount(raster rast, integer nband, boolean exclude\_nodata\_value, double precision searchvalue, double precision

roundto=0);

bigint ST\_ValueCount(raster rast, integer nband, double precision searchvalue, double precision roundto=0);

SETOF record ST\_ValueCount(text rastertable, text rastercolumn, integer nband=1, boolean exclude\_nodata\_value=true, dou-

ble precision[] searchvalues=NULL, double precision roundto=0, double precision OUT value, integer OUT count);

SETOF record ST\_ValueCount(text rastertable, text rastercolumn, double precision[] searchvalues, double precision roundto=0,

double precision OUT value, integer OUT count);

SETOF record ST\_ValueCount(text rastertable, text rastercolumn, integer nband, double precision[] searchvalues, double pre-

cision roundto=0, double precision OUT value, integer OUT count);

bigintST\_ValueCount(text rastertable, text rastercolumn, integer nband, boolean exclude\_nodata\_value, double precision search-

value, double precision roundto=0);

bigint ST\_ValueCount(text rastertable, text rastercolumn, double precision searchvalue, double precision roundto=0);

bigint ST\_ValueCount(text rastertable, text rastercolumn, integer nband, double precision searchvalue, double precision roundto=0);

**描述**

返回一个记录集，包括像素值和指定栅格（或栅格覆盖）的指定波段的像素值在一个值集合内的像素个数。如果没有指定波段，那么默认是波段1。如果没有指定searchvalues（满足条件的要搜索的像素值），那么会返回栅格所有的像素值统计。如果给定了searchvalue，那么会返回满足该值的所有像素的个数。

**注意**

如果exclude\_nodata\_value值为false，那么将会统计所有的像素个数。

可用版本：2.0.0

**样例**

UPDATE dummy\_rast SET rast = ST\_SetBandNoDataValue(rast,249) WHERE rid=2;

--样例 will count only pixels of band 1 that are not 249. --

SELECT (pvc).\*

FROM (SELECT ST\_ValueCount(rast) As pvc

FROM dummy\_rast WHERE rid=2) As foo

ORDER BY (pvc).value;

value | count

-------+-------

250 | 2

251 | 1

252 | 2

253 | 6

254 | 12

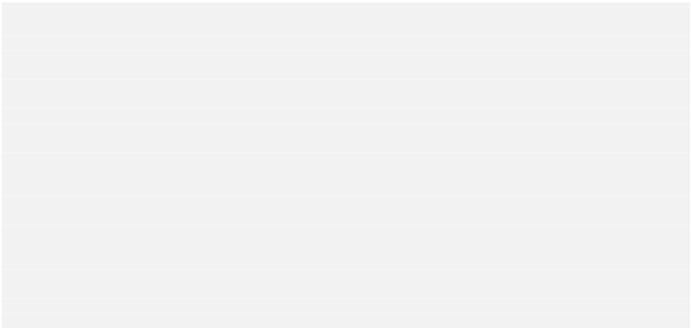
-- 样例 will coount all pixels of band 1 including 249 --

SELECT (pvc).\*

FROM (SELECT ST\_ValueCount(rast,1,false) As pvc

FROM dummy\_rast WHERE rid=2) As foo

ORDER BY (pvc).value;



PostGIS 2.2.0dev 开发手册

461 / 686

value | count

-------+-------

249 | 2

250 | 2

251 | 1

252 | 2

253 | 6

254 | 12

-- example will count only non-nodata value pixels of band 2

SELECT (pvc).\*

FROM (SELECT ST\_ValueCount(rast,2) As pvc

FROM dummy\_rast WHERE rid=2) As foo

ORDER BY (pvc).value;

value | count

-------+-------

78 | 1

79 | 1

88 | 1

89 | 1

96 | 1

97 | 1

98 | 1

99 | 2

112 | 2

:

--real live example.

Count all the pixels in an aerial raster tile band 2 intersecting a

←

geometry

-- and return only the pixel band values that have a count > 500

SELECT (pvc).value, SUM((pvc).count) As total

FROM (SELECT ST\_ValueCount(rast,2) As pvc

FROM o\_4\_boston

WHERE ST\_Intersects(rast,

ST\_GeomFromText(’POLYGON((224486 892151,224486 892200,224706 892200,224706

892151,224486 892151))’,26986)

)

) As foo

GROUP BY (pvc).value

HAVING SUM((pvc).count) > 500

ORDER BY (pvc).value;

value | total

-------+-----

51 | 502

54 | 521

-- Just return count of pixels in each raster tile that have value of 100 of tiles that

←

←

intersect

a specific geometry --

SELECT rid, ST\_ValueCount(rast,2,100) As count

FROM o\_4\_boston

WHERE ST\_Intersects(rast,

ST\_GeomFromText(’POLYGON((224486 892151,224486 892200,224706 892200,224706

892151,224486 892151))’,26986)

) ;

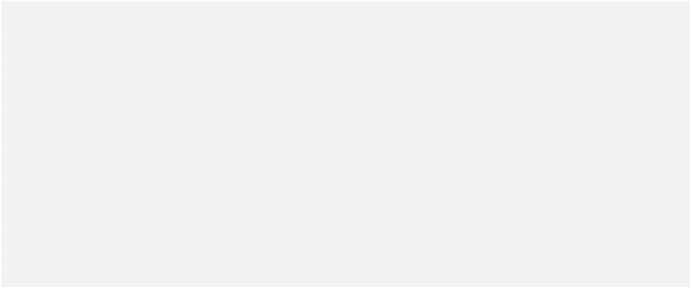
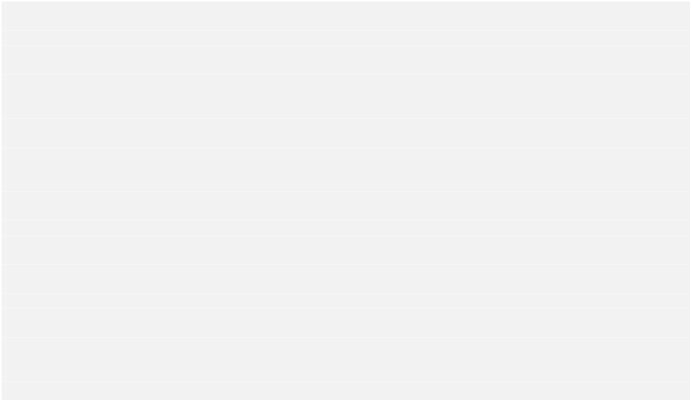
rid | count

-----+-------

1 | 56

2 | 95

←



PostGIS 2.2.0dev 开发手册

462 / 686

14 |

15 |

37

64

**相关请参考**

ST\_Count, ST\_SetBandNoDataValue

**9.10**

**9.10.1**

**栅格输出**

**ST\_AsBinary**

ST\_AsBinary — 返回不带要SRID元数据信息的WKB描述的栅格。

**用法**

bytea ST\_AsBinary(raster rast, boolean outasin=FALSE);

**描述**

返回不带要SRID元数据信息的WKB描述的栅格。如果参数outasin 值为TRUE,数据库之外的波段会被当做数据库内部的波段。对于使用binary 类型游标来获取数据库之外的数据来说，使用该参数不用将数据转换成文本表述，这很有用。

**注意**

默认情况下，WKB输出格式包含了数据库之外的外部文件路径。如果客户端没有访问数据库之外的栅格文件权限，那么把参数outasin值设置为TRUE。

版本提升： 2.1.0添加了参数outasin支持

**样例**

SELECT ST\_AsBinary(rast) As rastbin FROM dummy\_rast WHERE rid=1;

rastbin

---------------------------------------------------------------------------------

\001\000\000\000\000\000\000\000\000\000\000\000@\000\000\000\000\000\000\010@\

000\000\000\000\000\000\340?\000\000\000\000\000\000\340?\000\000\000\000\000\00

0\000\000\000\000\000\000\000\000\000\000\012\000\000\000\012\000\024\000

**9.10.2**

**ST\_AsGDALRaster**

ST\_AsGDALRaster —返回指定GDAL栅格格式的栅格瓦片。栅格格式是你的编译库所支持的其中一种。使用ST\_GDALRasters()得到你的库所支持的格式列表。

**用法**

bytea ST\_AsGDALRaster(raster rast, text format, text[] options=NULL, integer srid=sameassource);



PostGIS 2.2.0dev 开发手册

463 / 686

**描述**

返回指定GDAL栅格格式的栅格瓦片。函数参数意义如下：

· format 输出格式控制。这是由你编译的libgdal库的驱动所决定的。一般来说，支持格式有：’JPEG’,’GTIff’, ’PNG’。使用函数ST\_GDALDrivers得到你的库所支持的格式列表。

· options text型数组，GDAL参数，可选项是format参数决定的，参考GDAL Raster format options(http://www.gdal.org/frmt\_various.html)

· srs 嵌入到图像中的信息，取自spatial\_ref\_sys 表中的proj4text 或 srtext字段

可用版本：2.0.0 -需要GDAL版本>= 1.6.0.

**JPEG Output 样例**

SELECT ST\_AsGDALRaster(rast, ’JPEG’) As rastjpg

FROM dummy\_rast WHERE rid=1;

SELECT ST\_AsGDALRaster(rast, ’JPEG’, ARRAY[’QUALITY=50’]) As rastjpg

FROM dummy\_rast WHERE rid=2;

**GTIFF Output 样例**

SELECT ST\_AsGDALRaster(rast, ’GTiff’) As rastjpg

FROM dummy\_rast WHERE rid=2;

-- Out GeoTiff with jpeg compression, 90% quality

SELECT ST\_AsGDALRaster(rast, ’GTiff’,

ARRAY[’COMPRESS=JPEG’, ’JPEG\_QUALITY=90’],

4269) As rasttiff

FROM dummy\_rast WHERE rid=2;

**相关请参考**

Section 5.3, ST\_GDALDrivers, ST\_SRID

**9.10.3**

**ST\_AsJPEG**

ST\_AsJPEG — 返回指定波段的栅格瓦片为JPEG格式文件。如果没有指定波段，或者指定了但波段为1或超过3，那么只使用第一个波段。若指定了3个波段，那么3个波段都会使用，并且映射到RGB色彩模式中。

**用法**

bytea ST\_AsJPEG(raster rast, text[] options=NULL);

bytea ST\_AsJPEG(raster rast, integer nband, integer quality);

bytea ST\_AsJPEG(raster rast, integer nband, text[] options=NULL);

bytea ST\_AsJPEG(raster rast, integer[] nbands, text[] options=NULL);

bytea ST\_AsJPEG(raster rast, integer[] nbands, integer quality);

PostGIS 2.2.0dev 开发手册

464 / 686

**描述**

返回指定波段的栅格瓦片为JPEG格式文件。如果你需要导出较少常用的栅格类型数据，使用函数ST\_AsGDALRaster。如果没有指定波段，或者指定了但波段为1或超过3，那么只使用第一个波段。若指定了3个波段，那么3个波段都会使用。这个函数系列有很多变体。相关参数说明如下：

· nband 用于单个波段导出

· nbands是用于导出的波段数组（注意对于JPEG格式来说，该参数值最大为3），波段的顺序是RGB。例如ARRAY[3,2,1]意味着把波段3映射成RED（红色），波段2映射成green（绿色），波段1映射成blue（蓝色）（译者注：RGB颜色模式就是红绿蓝，然后进行混合形成其他颜色）

· quality 数字从0到100，值越大，图像渲染的越好。

· options GDAL中的text型数组参数，用于JPGE (参考ST\_GDALDrivers函数用于创建JPEG的选项). 对于JPEG，可选项是

PROGRESSIVE ON 或 OFF ， QUALITY 范围是0到100，默认是75。参考 [GDAL Rasterformat options](http://www.gdal.org/frmt_various.html)（http://www.gdal.org/frmt\_various.html）获取细节。

可用版本：2.0.0 -需要GDAL版本>= 1.6.0.

**样例: Output**

-- output first 3 bands 75% quality

SELECT ST\_AsJPEG(rast) As rastjpg

FROM dummy\_rast WHERE rid=2;

-- output only first band as 90% quality

SELECT ST\_AsJPEG(rast,1,90) As rastjpg

FROM dummy\_rast WHERE rid=2;

-- output first 3 bands (but make band 2 Red, band 1 green, and band 3 blue, progressive

and 90% quality

SELECT ST\_AsJPEG(rast,ARRAY[2,1,3],ARRAY[’QUALITY=90’,’PROGRESSIVE=ON’]) As rastjpg

FROM dummy\_rast WHERE rid=2;

**相关请参考**

Section 5.3, ST\_GDALDrivers, ST\_AsGDALRaster, ST\_AsPNG, ST\_AsTIFF

←

**9.10.4**

**ST\_AsPNG**

ST\_AsPNG —返回指定波段的栅格瓦片为PNG格式文件。如果波段中有1,3或4个波段，但没有指定波段，那么会使用所有波段。如果栅格波段又2个或者多于4个波段，并且没有指定波段，那么只会使用1个波段。波段会被映射到RGB或RGBA模式中。

**用法**

bytea ST\_AsPNG(raster rast, text[] options=NULL);

bytea ST\_AsPNG(raster rast, integer nband, integer compression);

bytea ST\_AsPNG(raster rast, integer nband, text[] options=NULL);

bytea ST\_AsPNG(raster rast, integer[] nbands, integer compression);

bytea ST\_AsPNG(raster rast, integer[] nbands, text[] options=NULL);

PostGIS 2.2.0dev 开发手册

465 / 686

**描述**

返回指定栅格的波段为PNG格式文件。如果你需要导出较少常用的栅格类型数据，使用函数ST\_AsGDALRaster。如果没有指定波段，那么只导出前3个波段。这个函数系列有很多变体。如果没有指定SRID，那么就会使用参数rast栅格的SRID。相关参数说明如下：

· nband 用于单个波段导出

· nbands是用于导出的波段数组（注意对于PNG格式来说，该参数值最大为3），波段的顺序是RGB。例如ARRAY[3,2,1]意味着把波段3映射成RED（红色），波段2映射成green（绿色），波段1映射成blue（蓝色）（译者注：RGB颜色模式就是红绿蓝，然后进行混合形成其他颜色）

·compression数字从1到9，值越大，压缩率越高。

· options GDAL中的text型数组参数，用于PNG (参考ST\_GDALDrivers函数用于创建JPEG的选项). 对于PNG，可选项是ZLEVEL（花费在压缩上的时间量-默认是6），例如ARRAY[’ZLEVEL=9’]. WORLDFILE是不允许的，因为这个函数必须做两份输出。参考GDAL Raster format options获取更多细节

可用版本：2.0.0 -需要GDAL版本>= 1.6.0.

**样例**

SELECT ST\_AsPNG(rast) As rastpng

FROM dummy\_rast WHERE rid=2;

-- export the first 3 bands and map band 3 to Red, band 1 to Green, band 2 to blue

SELECT ST\_AsPNG(rast, ARRAY[3,1,2]) As rastpng

FROM dummy\_rast WHERE rid=2;

**相关请参考**

ST\_AsGDALRaster, ST\_ColorMap, ST\_GDALDrivers, Section 5.3

**9.10.5**

**ST\_AsTIFF**

ST\_AsTIFF —返回栅格指定波段为简单的TIFF图像文件，如果没有指定波段，那么会尝试使用所有的波段。

**用法**

bytea ST\_AsTIFF(raster rast, text[] options=”, integer srid=sameassource);

bytea ST\_AsTIFF(raster rast, text compression=”, integer srid=sameassource);

bytea ST\_AsTIFF(raster rast, integer[] nbands, text compression=”, integer srid=sameassource);

bytea ST\_AsTIFF(raster rast, integer[] nbands, text[] options, integer srid=sameassource);

**描述**

返回栅格指定波段为简单的TIFF图像文件，如果没有指定波段，那么会尝试使用所有的波段。这个函数是ST\_AsGDALRaster的包装。

如果你需要导出较少常用的栅格类型数据，使用函数ST\_AsGDALRaster。这个函数系列有很多变体。如果没有提供空间参考系统表srstext字段值，将会使用栅格的空间参考。相关参数说明如下：

· nbands是用于导出的波段数组（注意对于PNG格式来说，该参数值最大为3），波段的顺序是RGB。例如ARRAY[3,2,1]意味着把波段3映射成RED（红色），波段2映射成green（绿色），波段1映射成blue（蓝色）（译者注：RGB颜色模式就是红绿蓝，然后进行混合形成其他颜色）

PostGIS 2.2.0dev 开发手册

466 / 686

compression表示压缩方式——JPEG90（默认90%，或者其他比例），LZW, JPEG, DEFLATE9。

· options GDAL中的text型数组参数，用于GTiff (参考ST\_GDALDrivers函数用于创建GTiff的选项). 参考GDAL Raster format options获取更多细节

· srid 空间参考系统表spatial\_ref\_sys中栅格的SRID值，这个用于填充地理参考信息。

可用版本：2.0.0 -需要GDAL版本>= 1.6.0.

**样例: Use jpeg compression 90%**

SELECT ST\_AsTIFF(rast, ’JPEG90’) As rasttiff

FROM dummy\_rast WHERE rid=2;

**相关请参考**

ST\_GDALDrivers, ST\_AsGDALRaster, ST\_SRID

**9.11**

**9.11.1**

**9.11.1.1**

**栅格处理函数**

**地图代数函数**

**ST\_Clip**

ST\_Clip — 返回输入几何对象修改后的栅格。如果波段没有指定，那么会处理所有的波段。如果参数crop没有指定或者值为TRUE，那么输出的栅格会被修改。

**用法**

raster ST\_Clip(raster rast, integer[] nband, geometry geom, double precision[] nodataval=NULL, boolean crop=TRUE);

raster ST\_Clip(raster rast, integer nband, geometry geom, double precision nodataval, boolean crop=TRUE);

raster ST\_Clip(raster rast, integer nband, geometry geom, boolean crop);

raster ST\_Clip(raster rast, geometry geom, double precision[] nodataval=NULL, boolean crop=TRUE);

raster ST\_Clip(raster rast, geometry geom, double precision nodataval, boolean crop=TRUE);

raster ST\_Clip(raster rast, geometry geom, boolean crop);

**描述**

返回输入几何对象修改后的栅格。如果波段没有指定，那么会处理所有的波段。函数ST\_Clip函数返回的栅格必须对每一个修剪的面的每一个波段都有一个NODATA值。如果没有提供任何值，输入的栅格也没有定义NODATA值，那么返回的结果栅格的NODATA值将被设定为函数ST\_MinPossibleValue(ST\_BandPixelTypband))的返回值。当输入参数nodataval数组元素个数少于波段数时候，nodataval数组最后一个数组元素的值将被用于多出的剩余波段。如果nodataval数组的元素个数多于波段数，那么多出的nodataval元素值将会被忽略掉。所有的函数变体都接受一个nodataval数组，也接受单个值并指定用于每一个波段。如果参数crop没有被指定，默认将取值为true，意义是输出栅格将被削平到几何对象和栅格边界相交的地方。

如果输入参数crop值为false，返回的新栅格将和输入栅格rast有相同的边界。

可用版本：2.0.0

版本提升： 2.1.0 Rewritten in C

样例 here use Massachusetts aerial data available on MassGIS site MassGIS Aerial Orthos. Coordinates are in Mas-

sachusetts State Plane Meters.

PostGIS 2.2.0dev 开发手册

467 / 686

**样例: 1 band clipping**

-- Clip the first band of an aerial tile by a 20 meter buffer.

SELECT ST\_Clip(rast, 1,

ST\_Buffer(ST\_Centroid(ST\_Envelope(rast)),20)

) from aerials.boston

WHERE rid = 4;

-- Demonstrate effect of crop on final dimensions of raster

-- 注意 how final extent is clipped to that of the geometry

-- if crop = true

SELECT ST\_XMax(ST\_Envelope(ST\_Clip(rast, 1, clipper, true))) As xmax\_w\_trim,

ST\_XMax(clipper) As xmax\_clipper,

ST\_XMax(ST\_Envelope(ST\_Clip(rast, 1, clipper, false))) As xmax\_wo\_trim,

ST\_XMax(ST\_Envelope(rast)) As xmax\_rast\_orig

FROM (SELECT rast, ST\_Buffer(ST\_Centroid(ST\_Envelope(rast)),6) As clipper

FROM aerials.boston

WHERE rid = 6) As foo;

xmax\_w\_trim

|

xmax\_clipper

|

xmax\_wo\_trim

|

xmax\_rast\_orig

------------------+------------------+------------------+------------------

230657.436173996 | 230657.436173996 | 230666.436173996 | 230666.436173996

Full raster tile before clipping

After Clipping

**样例: 1 band clipping with no crop and add back other bands unchanged**

-- Same example as before, but we need to set crop to false to be able to use ST\_AddBand

-- because ST\_AddBand requires all bands be the same Width and height

SELECT ST\_AddBand(ST\_Clip(rast, 1,

ST\_Buffer(ST\_Centroid(ST\_Envelope(rast)),20),false

), ARRAY[ST\_Band(rast,2),ST\_Band(rast,3)] ) from aerials.boston

WHERE rid = 6;



PostGIS 2.2.0dev 开发手册

468 / 686

Full raster tile before clipping

After Clipping - surreal

**样例: Clip all bands**

-- Clip all bands of an aerial tile by a 20 meter buffer.

-- Only difference is we don’t specify a specific band to clip

-- so all bands are clipped

SELECT ST\_Clip(rast,

ST\_Buffer(ST\_Centroid(ST\_Envelope(rast)), 20),

false

) from aerials.boston

WHERE rid = 4;

Full raster tile before clipping

**相关请参考**

ST\_AddBand, ST\_MapAlgebra, ST\_Intersection

After Clipping



PostGIS 2.2.0dev 开发手册

469 / 686

**9.11.1.2**

**ST\_ColorMap**

ST\_ColorMap —根据输入的源栅格和一个指定的波段，创建一个新的最多4个8BUI（grayscale，RGB,RGBA模式）波段的栅格。如果没有指定波段，那么默认为波段1。

**用法**

raster ST\_ColorMap(raster rast, integer nband=1, text colormap=grayscale, text method=INTERPOLATE);

raster ST\_ColorMap(raster rast, text colormap, text method=INTERPOLATE);

**描述**

把颜色表映射到一个栅格的波段位置为输入参数nband的波段，生成一个最多由4个8BUI波段组成的新栅格。新栅格的8BUI波段数是由颜色表的色彩组成类别数决定的。如果没有指定nband的值，那么默认取值为1。参数colormap可以是colormap预定于的关键字或者颜色组成的字符串。有效的预定义颜色关键字如下：

· grayscale 或 greyscale ：用于定义包含一个灰度8BUI波段的栅格。

· pseudocolor ：用于定义包含从蓝色到绿色到红色渐变的4个8BUI(RGBA)波段的栅格。

· fire ：用于定义包含从黑色到红色到浅黄色渐变的4个8BUI(RGBA)波段的栅格。

· bluered ：用于定义包含从蓝色到浅黄色到红色渐变的4个8BUI(RGBA)波段的栅格。

用户可以传递多行记录（每一行表示一条记录）的颜色值来自定义colormap参数。每一条记录通常包含5个值：像素值和对应的Red, Green, Blue, Alpha 成分 (颜色成分值在0到255范围内)。百分比也可以用于代替像素值，范围在0%到100%内，分别是栅格波段中最小和最大的值。每一行值可以用如下分隔符进行分隔：逗号(’,’), TAB符号, 冒号 (’:’) 和/或空格符号。像素值可以被设定为nv, null 或 nodata。一个样例如下：

5 0 0 0 255

4 100:50 55 255

1 150,100 150 255

0% 255 255 255 255

nv 0 0 0 0

colormap参数的语法和GDAL定义的颜色值语法相似GDAL [gdaldem.](http://www.gdal.org/gdaldem.html#gdaldem_color_relief)

可选有效的方法关键字如下：

· INTERPOLATE：使用线性插值法，把给定像素值范围内的颜色进行平滑变化。

· EXACT：严格匹配输入参数colormap中的像素值，如果像素值不匹配提供的colormap，那么像素值会被设定为0 0 0 0（RGBA）

· NEAREST：使用那些最贴近像素值的那些colormap记录。

**注意**

一个很不错的颜色表请参考： [ColorBrewer.](http://www.colorbrewer2.org)



PostGIS 2.2.0dev 开发手册

470 / 686

**警告**

返回的新栅格波段将不会设定NODATA值，如果需要设定这个值，使用函数ST\_SetBandNoDataValue进行设定

可用版本：2.1.0

**样例**

This is a junk table to play with

-- setup test raster table --

DROP TABLE IF EXISTS funky\_shapes;

CREATE TABLE funky\_shapes(rast raster);

INSERT INTO funky\_shapes(rast)

WITH ref AS (

SELECT ST\_MakeEmptyRaster( 200, 200, 0, 200, 1, -1, 0, 0) AS rast

)

SELECT

ST\_Union(rast)

FROM (

SELECT

ST\_AsRaster(

ST\_Rotate(

ST\_Buffer(

ST\_GeomFromText(’LINESTRING(0 2,50 50,150 150,125 50)’),

i\*2

),

pi() \* i \* 0.125, ST\_Point(50,50)

),

ref.rast, ’8BUI’::text, i \* 5

) AS rast

FROM ref

CROSS JOIN generate\_series(1, 10, 3) AS i

) AS shapes;

SELECT

ST\_NumBands(rast) As n\_orig,

ST\_NumBands(ST\_ColorMap(rast,1,

ST\_NumBands(ST\_ColorMap(rast,1,

ST\_NumBands(ST\_ColorMap(rast,1,

ST\_NumBands(ST\_ColorMap(rast,1,

ST\_NumBands(ST\_ColorMap(rast,1,

’greyscale’)) As ngrey,

’pseudocolor’)) As npseudo,

’fire’)) As nfire,

’bluered’)) As nbluered,

’

100% 255 0 0

80% 160 0 0

50% 130 0 0

30% 30 0 0

20% 60 0 0

0% 0 0 0

nv 255 255 255

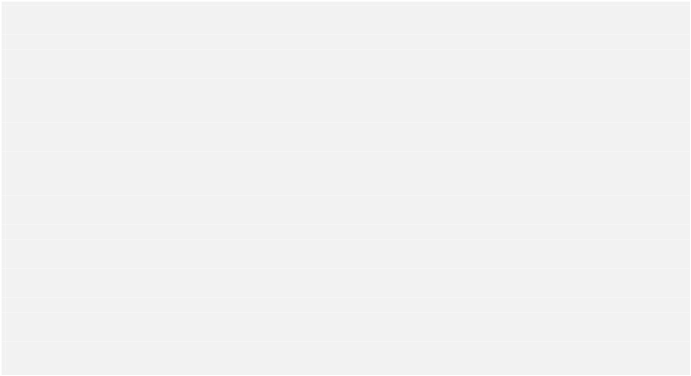
’)) As nred

FROM funky\_shapes;

n\_orig | ngrey | npseudo | nfire | nbluered | nred

--------+-------+---------+-------+----------+------

1 | 1 | 4 | 4 | 4 | 3



PostGIS 2.2.0dev 开发手册

471 / 686

**样例: Compare different color map looks using ST\_AsPNG**

SELECT

ST\_AsPNG(rast) As orig\_png,

ST\_AsPNG(ST\_ColorMap(rast,1,’greyscale’)) As grey\_png,

ST\_AsPNG(ST\_ColorMap(rast,1, ’pseudocolor’)) As pseudo\_png,

ST\_AsPNG(ST\_ColorMap(rast,1, ’nfire’)) As fire\_png,

ST\_AsPNG(ST\_ColorMap(rast,1, ’bluered’)) As bluered\_png,

ST\_AsPNG(ST\_ColorMap(rast,1, ’

100% 255 0 0

80% 160 0 0

50% 130 0 0

30% 30 0 0

20% 60 0 0

0% 0 0 0

nv 255 255 255

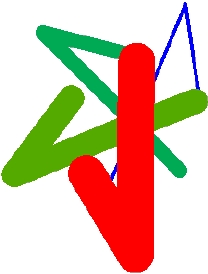
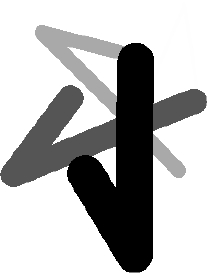
’)) As red\_png

FROM funky\_shapes;

orig\_png

grey\_png

pseudo\_png



PostGIS 2.2.0dev 开发手册

472 / 686

ﬁre\_png

bluered\_png

red\_png

**相关请参考**

ST\_AsPNG, ST\_AsRaster ST\_MapAlgebra, ST\_NumBands, ST\_Reclass, ST\_SetBandNoDataValue, ST\_Union

**9.11.1.3**

**ST\_Intersection**

ST\_Intersection —返回一个栅格或者几何对象-像素值键值对的集合，返回结果表示两个栅格共同部分或者栅格矢量化后与几何对象相交的结果。

**用法**

setof geomval ST\_Intersection(geometry geom, raster rast, integer band\_num=1);

setof geomval ST\_Intersection(raster rast, geometry geom);

setof geomval ST\_Intersection(raster rast, integer band\_num, geometry geom);

raster ST\_Intersection(raster rast1, raster rast2, double precision[] nodataval);

raster ST\_Intersection(raster rast1, raster rast2, text returnband=’BOTH’, double precision[] nodataval=NULL);

raster ST\_Intersection(raster rast1, integer band\_num1, raster rast2, integer band\_num2, double precision[] nodataval);

raster ST\_Intersection(raster rast1, integer band\_num1, raster rast2, integer band\_num2, text returnband=’BOTH’, double pre-

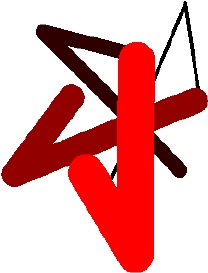
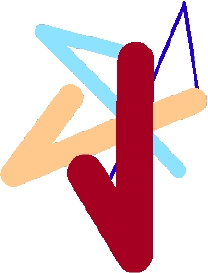
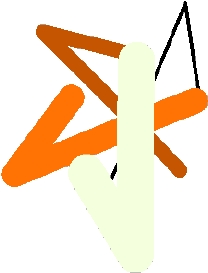
cision[] nodataval=NULL);

**描述**

返回一个栅格或者几何对象-像素值键值对的集合，返回结果表示两个栅格共同部分或者栅格矢量化后与几何对象相交的结果。

前3个函数变体，返回一个geomval集合，用于矢量空间处理。栅格首先（使用ST\_DumpAsPolygon函数）矢量化成一个geomval记录集合。然后这些记录再使用函数ST\_Intersection(geometry, geometry) 来进行交集处理。与一个栅格包含NODATA值的面相交的几何对象返回一个空的几何对象。通常这样的记录可以通过使用ST\_Intersect函数并在WHERE子句中过滤掉。你可以通过在表达式末尾添加.geom或.val的方式获取到几何对象和geomval记录集的数值部分，例如 (ST\_Intersection(rast, geom)).geom

其他函数变体返回一个栅格，适用于栅格空间。他们使用两个输入栅格的函数ST\_MapAlgebraExpr 来进行交集。



PostGIS 2.2.0dev 开发手册

473 / 686

返回栅格的边界是两个栅格边界进行几何相交的部分。返回栅格包含波段“BAND1”,"BAND2"或“BOTH”，这是由参数returnband传递什么值决定的。输入参数中任意波段中的NODATA值的面在返回结果中每个波段的面依然是NODATA值。

换句话说，任何像素与一个NODATA值的像素相交得到的结果是一个NODATA值的像素。

ST\_Intersection函数返回的栅格必须为没有相交的栅格平面指定一个NODATA值。你可以通过提供输入参数nodataval[]数组的值来定义或替换返回波段的nodata值，nodataval[]数组可以有1个或2个nodata 值，具体是几个取决于你想要下面那些波段：’BAND1’, ’BAND2’ 或’BOTH’波段。数组的第一个值替换第一个波段的nodata值，第二个值替换第二个波段的nodata值。如果一个输入的波段没有nodata值，并且没有提供nodataval数组值，那么会使用函数ST\_MinPossibleValue来确认nodata值。所有的函数变体都接受一个nodata值的数组也可以接受用于指定单个波段的单个值。

上面的所有函数变体，如果没有指定波段，那么默认使用波段1。

**注意**

想要更多控制返回结果的边界或者关于遇到一个nodata值时返回什么，请参考输入参数是两个栅格的函数ST\_MapAlgebraExpr.

**注意**

想要计算一个栅格波段和一个几何对象在栅格空间上的交集，使用函数ST\_Clip。ST\_Clip函数适用于多波段的栅格，但不返回对应栅格化几何对象的波段。

**注意**

ST\_Intersection函数应该和函数ST\_Intersects一起使用，栅格列和几何对象的列上都应该创建索引。

版本提升： 2.0.0 - 引入对栅格空间交集的支持。在2.0.0之前的版本，只支持矢量空间内的对象相交。

**样例: Geometry, Raster -- resulting in geometry vals**

SELECT

foo.rid,

foo.gid,

ST\_AsText((foo.geomval).geom) As geomwkt,

(foo.geomval).val

FROM (

SELECT

A.rid,

g.gid,

ST\_Intersection(A.rast, g.geom) As geomval

FROM dummy\_rast AS A

CROSS JOIN (

VALUES

(1, ST\_Point(3427928, 5793243.85) ),

(2, ST\_GeomFromText(’LINESTRING(3427927.85 5793243.75,3427927.8 5793243.75,3427927.8

5793243.8)’)),

(3, ST\_GeomFromText(’LINESTRING(1 2, 3 4)’))

) As g(gid,geom)

WHERE A.rid = 2

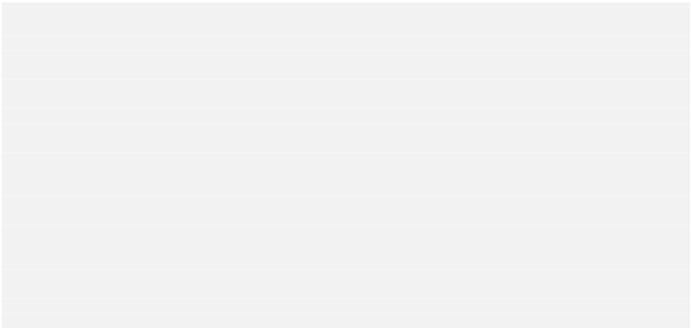
) As foo;

←

rid | gid |

geomwkt

| val



PostGIS 2.2.0dev 开发手册

474 / 686

-----+-----+-----------------------------------------------------------------------------------------

2

2

2

2

2

2

2

2

|

|

|

|

|

|

|

|

1

1

2

2

2

2

2

3

|

|

|

|

|

|

|

|

POINT(3427928 5793243.85) | 249

POINT(3427928 5793243.85) | 253

POINT(3427927.85 5793243.75) | 254

POINT(3427927.8 5793243.8) | 251

POINT(3427927.8 5793243.8) | 253

LINESTRING(3427927.8 5793243.75,3427927.8 5793243.8) | 252

MULTILINESTRING((3427927.8 5793243.8,3427927.8 5793243.75),...) | 250

GEOMETRYCOLLECTION EMPTY

**样例: Raster, Geometry -- resulting is a raster**

样例很快就有

**相关请参考**

geomval, ST\_Intersects, ST\_MapAlgebraExpr, ST\_Clip, ST\_AsText

**9.11.1.4**

**ST\_MapAlgebra**

ST\_MapAlgebra — 回调函数的版本——根据输入的一个或多个栅格，波段位置和用户指定的回调函数，返回一个单波段的栅格。

**用法**

raster ST\_MapAlgebra(rastbandarg[] rastbandargset, regprocedure callbackfunc, text pixeltype=NULL, text extenttype=INTERSECTIO

raster customextent=NULL, integer distancex=0, integer distancey=0, text[] VARIADIC userargs=NULL);

raster ST\_MapAlgebra(raster rast, integer[] nband, regprocedure callbackfunc, text pixeltype=NULL, text extenttype=FIRST,

raster customextent=NULL, integer distancex=0, integer distancey=0, text[] VARIADIC userargs=NULL);

raster ST\_MapAlgebra(raster rast, integer nband, regprocedure callbackfunc, text pixeltype=NULL, text extenttype=FIRST,

raster customextent=NULL, integer distancex=0, integer distancey=0, text[] VARIADIC userargs=NULL);

raster ST\_MapAlgebra(raster rast1, integer nband1, raster rast2, integer nband2, regprocedure callbackfunc, text pixeltype=NULL,

text extenttype=INTERSECTION, raster customextent=NULL, integer distancex=0, integer distancey=0, text[] VARIADIC user-

args=NULL);

**描述**

回调函数的版本——根据输入的一个或多个栅格，波段位置和用户指定的回调函数，返回一个单波段的栅格。

rast,rast1,rast2, rastbandargset ：表示需要在地图代数处理过程中评估、计算的栅格

rastbandargset：参数允许地图代数算子在多栅格和/或多波段上进行使用。参考函数变体1的样例：

nband, nband1, nband2 ：需要进行评估计算的栅格的波段数， nband可以是表示波段位置的整型或integer[]数组

对于2个栅格**/**2个波段的场景，nband1是栅格rast1的波段， nband2 是栅格rast2上的波段

**callbackfunc**：参数callbackfunc必须是SQL或SQL or PL/pgSQL 函数名，并且会被转换成regprocedure类型。

PL/pgSQL 函数的样例如下：

CREATE OR REPLACE FUNCTION sample\_callbackfunc(value double precision[][][], position

integer[][], VARIADIC userargs text[])

RETURNS double precision

AS $$

BEGIN

←

PostGIS 2.2.0dev 开发手册

475 / 686

RETURN 0;

END;

$$ LANGUAGE ’plpgsql’ IMMUTABLE;

参数callbackfunc 必须有3个参数: 一个3维双精度数组，一个2维整型数组和一个variadic（可变的，pg的数组类型关键字）类型的1维text数组。第一个参数值是所有输入栅格的值集合。这3维是（下标从1开始）：raster #, row y, column x。

第二个是参数的位置是输出栅格和输入栅格的像素集的位置。

外维（下标从0开始）是参数raster #（译者注：即第一个参数3维数组的第一个元素）

外维（译者注：内维和外维都是分形几何的概念，可以参考相关书籍）下标0的位置是输出栅格的像素位置。对于每一个外维度，在每一个内维度的X和Y都有两个元素。第三个参数用于传递用户任意指定的参数。

传递一个regprocedure类型参数给一个SQL函数需要传递一个完整的函数名标识，然后转换成regprocedure类型。例如传递上面样例函数作为回调函数使用方法如下：

’sample\_callbackfunc(double precision[], integer[], text[])’::regprocedure

注意这个回调函数参数包含了回调函数的函数名，函数参数类型，包围函数名和参数类型的单引号以及进行regprocedure的转换。

**pixeltype**：如果传递了参数pixeltype，新栅格的波段将会是pixeltype类型。如果pixeltype传递为NULL值或省略了，新栅格的波段将会和第一个参数栅格的指定波段有相同的pixeltype类型 (对于边界类型有:INTERSECTION, UNION, FIRST, CUSTOM) 或者合适栅格的指定波段（其边界类型为：SECOND，LAST）。如果对此有所疑问，那么在使用该函数就总是指定pixeltype类型好了。

输出栅格的pixeltype值必须是函数ST\_BandPixelType值的一种，或者省略或者设定为NULL

**extenttype**可能的值是INTERSECTION (默认值), UNION, FIRST (对于单个栅格变量的函数变体，取这个默认值), SECOND, LAST,CUSTOM.

**customextent**：如果参数extentype 值是CUSTOM，必须要为参数customextent提供一个参数。参数函数变体1的样例4。

**distancex**：参考栅格单元的以像素个数为单位计算的x方向距离。因此返回矩阵的宽度应该是2\*distancex + 1。如果没有指定，将值考虑参考栅格单元本身（相邻的像素为0）

**distancey**：参考栅格单元的以像素个数为单位计算的y方向距离。因此返回矩阵的高度应该是2\*distancey + 1。如果没有指定，将值考虑参考栅格单元本身（相邻的像素为0）。

**userargs：**回调函数参数的第三个参数是一个variadic text型数组。所有的其他需要自定义添加的参数都通过这个参数来传递给指定的回调函数callbackfunc

**注意**

关于关键字VARIADIC的信息，请参考PostgreSQL文档的章节"SQL

Functions with Variable Numbers of Arguments" section of [Query Language (SQL) Functions.](http://www.postgresql.org/docs/current/static/xfunc-sql.html)

**注意**

回调函数必须要传递参数text[], 无论你选择传递任意的一个参数给callbackfunction是否要进行处理

函数变体1接受一个rastbandarg数组，参数允许地图代数算子在多栅格和/或多波段上进行使用

参考函数变体1的样例。

函数变体2和3可以操作一个栅格的一个或多个波段。参考函数变体2和3的样例。

函数变体4操作两个栅格并且每个栅格都有一个波段，参考函数变体4的样例。

可用版本：2.1.0



PostGIS 2.2.0dev 开发手册

476 / 686

**样例: Variant 1**

One raster, one band

WITH foo AS (

SELECT 1 AS rid, ST\_AddBand(ST\_MakeEmptyRaster(2, 2, 0, 0, 1, -1, 0, 0, 0), 1, ’16BUI’,

1, 0) AS rast

)

SELECT

ST\_MapAlgebra(

ARRAY[ROW(rast, 1)]::rastbandarg[],

’sample\_callbackfunc(double precision[], int[], text[])’::regprocedure

) AS rast

FROM foo

One raster, several bands

WITH foo AS (

SELECT 1 AS rid, ST\_AddBand(ST\_AddBand(ST\_AddBand(ST\_MakeEmptyRaster(2, 2, 0, 0, 1, -1,

0, 0, 0), 1, ’16BUI’, 1, 0), 2, ’8BUI’, 10, 0), 3, ’32BUI’, 100, 0) AS rast

)

SELECT

ST\_MapAlgebra(

ARRAY[ROW(rast, 3), ROW(rast, 1), ROW(rast, 3), ROW(rast, 2)]::rastbandarg[],

’sample\_callbackfunc(double precision[], int[], text[])’::regprocedure

) AS rast

FROM foo

Several rasters, several bands

WITH foo AS (

SELECT 1 AS rid, ST\_AddBand(ST\_AddBand(ST\_AddBand(ST\_MakeEmptyRaster(2, 2, 0, 0, 1, -1,

0, 0, 0), 1, ’16BUI’, 1, 0), 2, ’8BUI’, 10, 0), 3, ’32BUI’, 100, 0) AS rast UNION ALL

SELECT 2 AS rid, ST\_AddBand(ST\_AddBand(ST\_AddBand(ST\_MakeEmptyRaster(2, 2, 0, 1, 1, -1,

←

←

←

←

0, 0, 0), 1, ’16BUI’, 2, 0), 2, ’8BUI’, 20, 0), 3, ’32BUI’, 300, 0) AS rast

)

SELECT

ST\_MapAlgebra(

ARRAY[ROW(t1.rast, 3), ROW(t2.rast, 1), ROW(t2.rast, 3), ROW(t1.rast, 2)]::rastbandarg ←

[],

’sample\_callbackfunc(double precision[], int[], text[])’::regprocedure

) AS rast

FROM foo t1

CROSS JOIN foo t2

WHERE t1.rid = 1

AND t2.rid = 2

Complete example of tiles of a coverage with neighborhood. This query only works with PostgreSQL 9.1 or higher.

WITH foo AS (

SELECT 0 AS rid, ST\_AddBand(ST\_MakeEmptyRaster(2, 2, 0, 0, 1, -1, 0, 0, 0), 1, ’16BUI’,

←

1, 0) AS rast UNION ALL

SELECT 1, ST\_AddBand(ST\_MakeEmptyRaster(2, 2, 2, 0, 1, -1, 0, 0, 0), 1, ’16BUI’, 2, 0) AS ←

rast UNION ALL

SELECT 2, ST\_AddBand(ST\_MakeEmptyRaster(2, 2, 4, 0, 1, -1, 0, 0, 0), 1, ’16BUI’, 3, 0) AS ←

rast UNION ALL

SELECT

AS

SELECT

AS

SELECT

AS

3, ST\_AddBand(ST\_MakeEmptyRaster(2, 2, 0, -2, 1, -1, 0, 0, 0), 1, ’16BUI’, 10, 0)

rast UNION ALL

4, ST\_AddBand(ST\_MakeEmptyRaster(2, 2, 2, -2, 1, -1, 0, 0, 0), 1, ’16BUI’, 20, 0)

rast UNION ALL

5, ST\_AddBand(ST\_MakeEmptyRaster(2, 2, 4, -2, 1, -1, 0, 0, 0), 1, ’16BUI’, 30, 0)

rast UNION ALL

←

←

←

PostGIS 2.2.0dev 开发手册

477 / 686

SELECT 6, ST\_AddBand(ST\_MakeEmptyRaster(2, 2, 0, -4, 1, -1, 0, 0, 0), 1, ’16BUI’, 100, 0) ←

AS rast UNION ALL

SELECT 7, ST\_AddBand(ST\_MakeEmptyRaster(2, 2, 2, -4, 1, -1, 0, 0, 0), 1, ’16BUI’, 200, 0) ←

AS rast UNION ALL

SELECT 8, ST\_AddBand(ST\_MakeEmptyRaster(2, 2, 4, -4, 1, -1, 0, 0, 0), 1, ’16BUI’, 300, 0) ←

AS rast

)

SELECT

t1.rid,

ST\_MapAlgebra(

ARRAY[ROW(ST\_Union(t2.rast), 1)]::rastbandarg[],

’sample\_callbackfunc(double precision[], int[], text[])’::regprocedure,

’32BUI’,

’CUSTOM’, t1.rast,

1, 1

) AS rast

FROM foo t1

CROSS JOIN foo t2

WHERE t1.rid = 4

AND t2.rid BETWEEN 0 AND 8

AND ST\_Intersects(t1.rast, t2.rast)

GROUP BY t1.rid, t1.rast

Examples like the prior one for tiles of a coverage with neighborhood but works with PostgreSQL 9.0.

WITH src AS (

SELECT 0 AS rid, ST\_AddBand(ST\_MakeEmptyRaster(2, 2, 0, 0, 1, -1, 0, 0, 0), 1, ’16BUI’,

←

1, 0) AS rast UNION ALL

SELECT 1, ST\_AddBand(ST\_MakeEmptyRaster(2, 2, 2, 0, 1, -1, 0, 0, 0), 1, ’16BUI’, 2, 0) AS ←

rast UNION ALL

SELECT 2, ST\_AddBand(ST\_MakeEmptyRaster(2, 2, 4, 0, 1, -1, 0, 0, 0), 1, ’16BUI’, 3, 0) AS ←

rast UNION ALL

SELECT

AS

SELECT

AS

SELECT

AS

3, ST\_AddBand(ST\_MakeEmptyRaster(2, 2, 0, -2, 1, -1, 0, 0, 0), 1, ’16BUI’, 10, 0)

rast UNION ALL

4, ST\_AddBand(ST\_MakeEmptyRaster(2, 2, 2, -2, 1, -1, 0, 0, 0), 1, ’16BUI’, 20, 0)

rast UNION ALL

5, ST\_AddBand(ST\_MakeEmptyRaster(2, 2, 4, -2, 1, -1, 0, 0, 0), 1, ’16BUI’, 30, 0)

rast UNION ALL

←

←

←

SELECT 6, ST\_AddBand(ST\_MakeEmptyRaster(2, 2, 0, -4, 1, -1, 0, 0, 0), 1, ’16BUI’, 100, 0) ←

AS rast UNION ALL

SELECT 7, ST\_AddBand(ST\_MakeEmptyRaster(2, 2, 2, -4, 1, -1, 0, 0, 0), 1, ’16BUI’, 200, 0) ←

AS rast UNION ALL

SELECT 8, ST\_AddBand(ST\_MakeEmptyRaster(2, 2, 4, -4, 1, -1, 0, 0, 0), 1, ’16BUI’, 300, 0) ←

AS rast

)

WITH foo AS (

SELECT

t1.rid,

ST\_Union(t2.rast) AS rast

FROM src t1

JOIN src t2

ON ST\_Intersects(t1.rast, t2.rast)

AND t2.rid BETWEEN 0 AND 8

WHERE t1.rid = 4

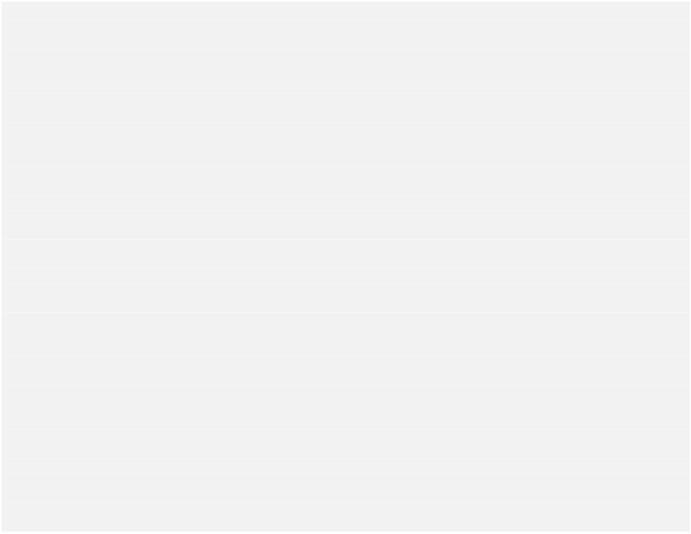
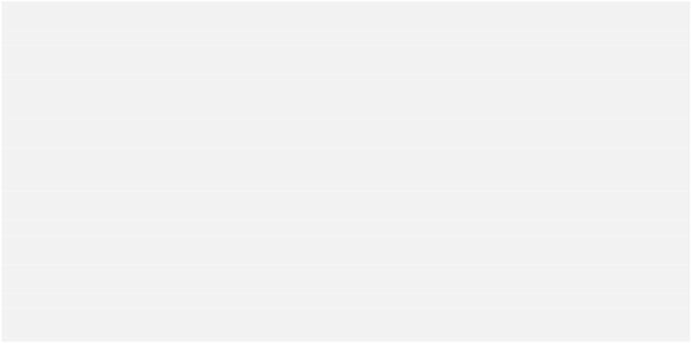
GROUP BY t1.rid

), bar AS (

SELECT

t1.rid,

ST\_MapAlgebra(



PostGIS 2.2.0dev 开发手册

478 / 686

ARRAY[ROW(t2.rast, 1)]::rastbandarg[],

’raster\_nmapalgebra\_test(double precision[], int[], text[])’::regprocedure,

’32BUI’,

’CUSTOM’, t1.rast,

1, 1

) AS rast

FROM src t1

JOIN foo t2

ON t1.rid = t2.rid

)

SELECT

rid,

(ST\_Metadata(rast)),

(ST\_BandMetadata(rast, 1)),

ST\_Value(rast, 1, 1, 1)

FROM bar;

**样例: Variants 2 and 3**

One raster, several bands

WITH foo AS (

SELECT 1 AS rid, ST\_AddBand(ST\_AddBand(ST\_AddBand(ST\_MakeEmptyRaster(2, 2, 0, 0, 1, -1,

0, 0, 0), 1, ’16BUI’, 1, 0), 2, ’8BUI’, 10, 0), 3, ’32BUI’, 100, 0) AS rast

)

SELECT

ST\_MapAlgebra(

rast, ARRAY[3, 1, 3, 2]::integer[],

’sample\_callbackfunc(double precision[], int[], text[])’::regprocedure

) AS rast

FROM foo

One raster, one band

WITH foo AS (

SELECT 1 AS rid, ST\_AddBand(ST\_AddBand(ST\_AddBand(ST\_MakeEmptyRaster(2, 2, 0, 0, 1, -1,

0, 0, 0), 1, ’16BUI’, 1, 0), 2, ’8BUI’, 10, 0), 3, ’32BUI’, 100, 0) AS rast

)

SELECT

ST\_MapAlgebra(

rast, 2,

’sample\_callbackfunc(double precision[], int[], text[])’::regprocedure

) AS rast

FROM foo

**样例: Variant 4**

Two rasters, two bands

WITH foo AS (

←

←

SELECT 1 AS rid,

0, 0, 0), 1,

SELECT 2 AS rid,

0, 0, 0), 1,

ST\_AddBand(ST\_AddBand(ST\_AddBand(ST\_MakeEmptyRaster(2,

’16BUI’, 1, 0), 2, ’8BUI’, 10, 0), 3, ’32BUI’, 100, 0)

ST\_AddBand(ST\_AddBand(ST\_AddBand(ST\_MakeEmptyRaster(2,

’16BUI’, 2, 0), 2, ’8BUI’, 20, 0), 3, ’32BUI’, 300, 0)

2,

AS

2,

AS

0, 0, 1, -1,

rast UNION ALL

0, 1, 1, -1,

rast

←

←

)

SELECT

ST\_MapAlgebra(

t1.rast, 2,

t2.rast, 1,

PostGIS 2.2.0dev 开发手册

479 / 686

’sample\_callbackfunc(double precision[], int[], text[])’::regprocedure

) AS rast

FROM foo t1

CROSS JOIN foo t2

WHERE t1.rid = 1

AND t2.rid = 2

**相关请参考**

rastbandarg, ST\_Union, ST\_MapAlgebra

**9.11.1.5**

**ST\_MapAlgebra**

ST\_MapAlgebra —带有SQL表达式参数expression的函数版本。根据一个或更多输入的栅格、波段以及一个或更多用户自定义的SQL表达式。

**用法**

raster ST\_MapAlgebra(raster rast, integer nband, text pixeltype, text expression, double precision nodataval=NULL);

raster ST\_MapAlgebra(raster rast, text pixeltype, text expression, double precision nodataval=NULL);

raster ST\_MapAlgebra(raster rast1, integer nband1, raster rast2, integer nband2, text expression, text pixeltype=NULL, text

extenttype=INTERSECTION, text nodata1expr=NULL, text nodata2expr=NULL, double precision nodatanodataval=NULL);

raster ST\_MapAlgebra(raster rast1, raster rast2, text expression, text pixeltype=NULL, text extenttype=INTERSECTION, text

nodata1expr=NULL, text nodata2expr=NULL, double precision nodatanodataval=NULL);

**描述**

带有SQL表达式参数expression的函数版本。根据一个或更多输入的栅格、波段以及一个或更多用户自定义的SQL表达式。

可用版本：2.1.0

**描述: 函数变体1和变体2（处理一个栅格的函数变体）**

根据加在输入栅格rast参数上的一个有效的PostgreSQLSQL表达式，创建一个新的单波段栅格。如果没有提供参数nband，那么指定为波段1。新栅格将会和原始的输入栅格有相同的空间参考，宽度和高度，但是只会包含一个波段。如果传递了参数pixeltype，那么新的栅格波段将会有一个波段的pixeltpye和传递参数值一样；如果传递的为NULL，那么新栅格的波段的pixeltype和输入栅格波段一样。

·表达式允许的关键字如下：

1. [rast] –感兴趣的像素的像素值

2. [rast.val] –感兴趣像素的像素值

3. [rast.x] –感兴趣的像素的下标从1开始的像素的列位置

4. [rast.y] -感兴趣的像素的下标从1开始的像素的行位置

PostGIS 2.2.0dev 开发手册

480 / 686

**描述: 函数变体3和4（应用于两个栅格）**

根据加在两个输入栅格rast1和rast2参数上的一个有效的PostgreSQLSQL表达式，创建一个新的单波段栅格。如果没有指定band1和band2参数，那么默认取波段1。返回的栅格对齐方式会按照第一个输入栅格的参数（scale, skew 和 pixel corners）进行对齐。返回栅格将会的边界是由extenttype参数定义的。

**expression：**一个PostgreSQL的算术表达式，用算术运算符把两个栅格连接起来，如果两个像素相交，那么像素值的定义可以如下：(([rast1] + [rast2])/2.0)::integer

**pixeltype**：定义输出栅格的像素类型（pixel type），该参数的值必须是定义在函数ST\_BandPixelType的值输出范围内的，定义时候也可以省略或者设置为NULL。如果没有传递该值或者定义位NULL，默认的像素类型和第一个输入栅格像素类型一致。

**extenttype**：控制输出栅格的边界结合方式（译者注：就是两个输入边界的怎样结合）

1. INTERSECTION –新栅格的边界是两个输入栅格的交集，这是默认的方式。

2. UNION –新栅格的边界是两个输入栅格的并集。

3. FIRST –新栅格的边界和第一个栅格的边界相同。

4. SECOND –新栅格的边界和第二个栅格的边界相同。

**nodata1expr**：一个只包含rast2或一个常量的代数表达式，用于定义当rast1栅格的像素都是NODATA值的，而rast2的像素却是有值的时候，该返回该表达式的值。

**nodata2expr**：一个只包含rast1或一个常量的代数表达式，用于定义当rast2栅格的像素都是NODATA值的，而rast1的像素却是有值的时候，该返回该表达式的值。

**nodatanodataval**：一个numeric常量，用于定义当rast1和rast2的像素都是NODATA值时候，返回这个常量。

·参数expression, nodata1expr 和 nodata2expr允许的关键字如下：

1. [rast1] –栅格rast1的感兴趣的像素的像素值

2. [rast1.val] -栅格rast1的感兴趣的像素的像素值

3. [rast1.x] –栅格rast1的感兴趣的像素的的列位置，下标从1开始

4. [rast1.y] -栅格rast1的感兴趣的像素的的行位置，下标从1开始

5. [rast2] -栅格rast2的感兴趣的像素的像素值

6. [rast2.val] -栅格rast2的感兴趣的像素的像素值

7. [rast2.x] -栅格rast2的感兴趣的像素的的列位置，下标从1开始

8. [rast2.y] -栅格rast2的感兴趣的像素的的行位置，下标从1开始

**样例: Variants 1 and 2**

WITH foo AS (

SELECT ST\_AddBand(ST\_MakeEmptyRaster(10, 10, 0, 0, 1, 1, 0, 0, 0), ’32BF’, 1, -1) AS rast

)

SELECT

ST\_MapAlgebra(rast, 1, NULL, ’ceil([rast]\*[rast.x]/[rast.y]+[rast.val])’)

FROM foo

PostGIS 2.2.0dev 开发手册

481 / 686

**样例: Variant 3 and 4**

WITH foo AS (

SELECT 1 AS rid, ST\_AddBand(ST\_AddBand(ST\_AddBand(ST\_MakeEmptyRaster(2,

0, 0, 0), 1, ’16BUI’, 1, 0), 2, ’8BUI’, 10, 0), 3, ’32BUI’, 100, 0)

SELECT 2 AS rid, ST\_AddBand(ST\_AddBand(ST\_AddBand(ST\_MakeEmptyRaster(2,

0, 0, 0), 1, ’16BUI’, 2, 0), 2, ’8BUI’, 20, 0), 3, ’32BUI’, 300, 0)

2,

AS

2,

AS

0, 0, 1, -1,

rast UNION ALL

0, 1, 1, -1,

rast

←

←

)

SELECT

ST\_MapAlgebra(

t1.rast, 2,

t2.rast, 1,

’([rast2] + [rast1.val]) / 2’

) AS rast

FROM foo t1

CROSS JOIN foo t2

WHERE t1.rid = 1

AND t2.rid = 2

**相关请参考**

rastbandarg, ST\_Union, ST\_MapAlgebra

**9.11.1.6**

**ST\_MapAlgebraExpr**

ST\_MapAlgebraExpr —输入参数是1个波段的函数版本。根据一个用户定义在输入栅格波段和pixeltype值的PostgreSQL代数表达式，生成一个带有一个波段的栅格。如果没有指定波段是哪个，默认取波段1。

**用法**

raster ST\_MapAlgebraExpr(raster rast, integer band, text pixeltype, text expression, double precision nodataval=NULL);

raster ST\_MapAlgebraExpr(raster rast, text pixeltype, text expression, double precision nodataval=NULL);

**描述**

**警告**

ST\_MapAlgebraExpr在2.1.0+版本不推荐使用，推荐使用函数ST\_MapAlgebra

根据一个用户定义在输入栅格波段和pixeltype值的PostgreSQL代数表达式，生成一个带有一个波段的栅格。如果没有指定波段是哪个，默认取波段1。新栅格将会和原始的输入栅格有相同的空间参考，宽度和高度，但是只会包含一个波段。

如果传递了参数pixeltype，那么新的栅格波段将会有一个波段的pixeltpye和传递参数值一样；

如果传递的为NULL，那么新栅格的波段的pixeltype和输入栅格波段一样。

在参数expression中你可以使用表达式 [rast] 来引用原始输入波段的像素值，使用表达式 [rast.x] 来引用下标从1开始的像素列的位置，使用表达式[rast.y]来引用下标从1开始的像素行的位置。

可用版本：2.0.0



PostGIS 2.2.0dev 开发手册

482 / 686

**样例**

Create a new 1 band raster from our original that is a function of modulo 2 of the original raster band.

ALTER TABLE dummy\_rast ADD COLUMN map\_rast raster;

UPDATE dummy\_rast SET map\_rast = ST\_MapAlgebraExpr(rast,NULL,’mod([rast],2)’) WHERE rid =

2;

SELECT

ST\_Value(rast,1,i,j) As origval,

ST\_Value(map\_rast, 1, i, j) As mapval

FROM dummy\_rast

CROSS JOIN generate\_series(1, 3) AS i

CROSS JOIN generate\_series(1,3) AS j

WHERE rid = 2;

origval | mapval

---------+--------

253 | 1

254 | 0

253 | 1

253 | 1

254 | 0

254 | 0

250 | 0

254 | 0

254 | 0

Create a new 1 band raster of pixel-type 2BUI from our original that is reclassiﬁed and set the nodata value to be 0.

ALTER TABLE dummy\_rast ADD COLUMN map\_rast2 raster;

UPDATE dummy\_rast SET

map\_rast2 = ST\_MapAlgebraExpr(rast,’2BUI’,’CASE WHEN [rast] BETWEEN 100 and 250 THEN 1

WHEN [rast] = 252 THEN 2 WHEN [rast] BETWEEN 253 and 254 THEN 3 ELSE 0 END’, ’0’)

WHERE rid = 2;

SELECT DISTINCT

ST\_Value(rast,1,i,j) As origval,

ST\_Value(map\_rast2, 1, i, j) As mapval

FROM dummy\_rast

CROSS JOIN generate\_series(1, 5) AS i

CROSS JOIN generate\_series(1,5) AS j

WHERE rid = 2;

origval | mapval

---------+--------

249 | 1

250 | 1

251 |

252 | 2

253 | 3

254 | 3

SELECT

ST\_BandPixelType(map\_rast2) As b1pixtyp

FROM dummy\_rast

WHERE rid = 2;

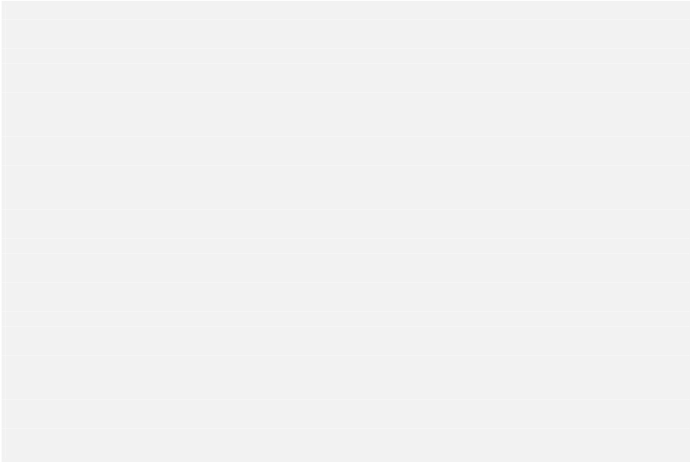
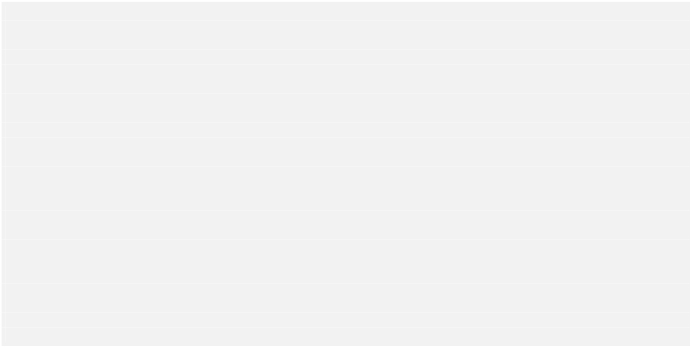
b1pixtyp

----------

2BUI

←

←



PostGIS 2.2.0dev 开发手册

483 / 686

original (column rast-view)

rast\_view\_ma

Create a new 3 band raster same pixel type from our original 3 band raster with ﬁrst band altered by map algebra and remaining

2 bands unaltered.

SELECT

ST\_AddBand(

ST\_AddBand(

ST\_AddBand(

ST\_MakeEmptyRaster(rast\_view),

ST\_MapAlgebraExpr(rast\_view,1,NULL,’tan([rast])\*[rast]’)

),

ST\_Band(rast\_view,2)

),

ST\_Band(rast\_view, 3) As rast\_view\_ma

)

FROM wind

WHERE rid=167;

**相关请参考**

ST\_MapAlgebraExpr, ST\_MapAlgebraFct, ST\_BandPixelType, ST\_GeoReference, ST\_Value

**9.11.1.7**

**ST\_MapAlgebraExpr**

ST\_MapAlgebraExpr —输入参数是两个栅格波段的版本。根据一个用户定义在两个输入参数栅格波段和pixeltype值的PostgreSQL代数表达式，生成一个带有一个波段的栅格。如果没有指定每一个栅格的波段是哪个，那么默认都取波段1。输出栅格的对其方式 (scale, skew and pixel corners) 和第一个栅格参数相同，边界定义由参数"extenttype"定义，参数"extenttype"的值可以是: INTERSECTION, UNION, FIRST, SECOND。

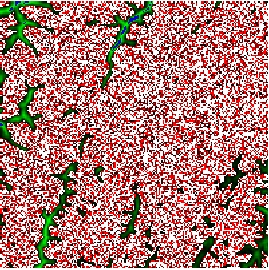
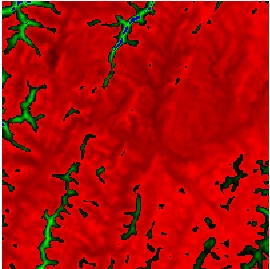
**用法**

raster ST\_MapAlgebraExpr(raster rast1, raster rast2, text expression, text pixeltype=same\_as\_rast1\_band, text extenttype=INTERSECT

text nodata1expr=NULL, text nodata2expr=NULL, double precision nodatanodataval=NULL);

raster ST\_MapAlgebraExpr(raster rast1, integer band1, raster rast2, integer band2, text expression, text pixeltype=same\_as\_rast1\_band,

text extenttype=INTERSECTION, text nodata1expr=NULL, text nodata2expr=NULL, double precision nodatanodataval=NULL);



PostGIS 2.2.0dev 开发手册

484 / 686

**描述**

**警告**

ST\_MapAlgebraExpr在2.1.0+版本不推荐使用，推荐使用函数ST\_MapAlgebra

输入参数是两个栅格波段的版本。根据一个用户定义在两个输入参数栅格波段和pixeltype值的PostgreSQL代数表达式，生成一个带有一个波段的栅格。如果没有指定每一个栅格的波段是哪个，那么默认都取波段1。输出栅格的对其方式 (scale, skew and pixel corners) 和第一个栅格参数相同，边界定义由参数"extenttype"定义，参数"extenttype"的值可以是: INTERSECTION, UNION, FIRST, SECOND。

**expression：**一个PostgreSQL的算术表达式，用算术运算符把两个栅格连接起来，如果两个像素相交，那么像素值的定义可以如下：(([rast1] + [rast2])/2.0)::integer

**pixeltype**：定义输出栅格的像素类型（pixel type），该参数的值必须是定义在函数ST\_BandPixelType的值输出范围内的，定义时候也可以省略或者设置为NULL。如果没有传递该值或者定义位NULL，默认的像素类型和第一个输入栅格像素类型一致。

**extenttype**：控制输出栅格的边界结合方式（译者注：就是两个输入边界的怎样结合）

1. INTERSECTION –新栅格的边界是两个输入栅格的交集，这是默认的方式。

2. UNION – 新栅格的边界是两个输入栅格的并集。

3. FIRST – 新栅格的边界和第一个栅格的边界相同。

4. SECOND – 新栅格的边界和第二个栅格的边界相同。

**nodata1expr**：一个只包含rast2或一个常量的代数表达式，用于定义当rast1栅格的像素都是NODATA值的，而rast2的像素却是有值的时候，该返回该表达式的值。

**nodata2expr**：一个只包含rast1或一个常量的代数表达式，用于定义当rast2栅格的像素都是NODATA值的，而rast1的像素却是有值的时候，该返回该表达式的值。

**nodatanodataval**：一个numeric常量，用于定义当rast1和rast2的像素都是NODATA值时候，返回这个常量。

如果传递了pixeltype参数值，那么新栅格将会有一个波段的像素值和传递的参数pixeltype值一样。如果传递的是NULL或者没有指定pixeltype值，那么新栅格的波段将会和输入的栅格rast1的波段的pixeltype值一致。

使用表达式 [rast1.val] [rast2.val] 来引用原始栅格波段的像素值，使用[rast1.x], [ras-

t1.y] 来引用像素的行号和列号位置

可用版本：2.0.0

**样例: 2 Band Intersection and Union**

Create a new 1 band raster from our original that is a function of modulo 2 of the original raster band.

--Create a cool set of rasters --

DROP TABLE IF EXISTS fun\_shapes;

CREATE TABLE fun\_shapes(rid serial PRIMARY KEY, fun\_name text, rast raster);

-- Insert some cool shapes around Boston in Massachusetts state plane meters --

INSERT INTO fun\_shapes(fun\_name, rast)

VALUES (’ref’, ST\_AsRaster(ST\_MakeEnvelope(235229, 899970, 237229, 901930,26986),200,200,’8 ←

BUI’,0,0));

INSERT INTO fun\_shapes(fun\_name,rast)

WITH ref(rast) AS (SELECT rast FROM fun\_shapes WHERE fun\_name = ’ref’ )

SELECT ’area’ AS fun\_name, ST\_AsRaster(ST\_Buffer(ST\_SetSRID(ST\_Point(236229, 900930),26986) ←

, 1000),

ref.rast,’8BUI’, 10, 0) As rast



PostGIS 2.2.0dev 开发手册

485 / 686

FROM ref

UNION ALL

SELECT ’rand bubbles’,

ST\_AsRaster(

(SELECT ST\_Collect(geom)

FROM (SELECT ST\_Buffer(ST\_SetSRID(ST\_Point(236229 + i\*random()\*100, 900930 + j\*random() ←

\*100),26986), random()\*20) As geom

FROM generate\_series(1,10) As i, generate\_series(1,10) As j

) As foo ), ref.rast,’8BUI’, 200, 0)

FROM ref;

--map them -

SELECT

ST\_MapAlgebraExpr(

area.rast, bub.rast, ’[rast2.val]’, ’8BUI’, ’INTERSECTION’, ’[rast2.val]’, ’[rast1.val ←

]’) As interrast,

ST\_MapAlgebraExpr(

area.rast, bub.rast, ’[rast2.val]’, ’8BUI’, ’UNION’, ’[rast2.val]’, ’[rast1.val]’) As ←

unionrast

FROM

(SELECT rast FROM fun\_shapes WHERE

fun\_name = ’area’) As area

CROSS JOIN

(SELECT rast

FROM fun\_shapes WHERE

fun\_name = ’rand bubbles’) As bub

mapalgebra intersection

map algebra union

**样例: Overlaying rasters on a canvas as separate bands**

-- we use ST\_AsPNG to render the image so all single band ones look grey --

WITH mygeoms

AS ( SELECT 2 As bnum, ST\_Buffer(ST\_Point(1,5),10) As geom

UNION ALL

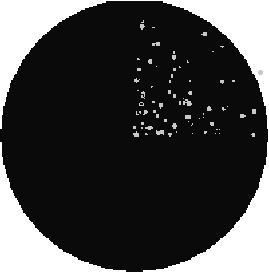
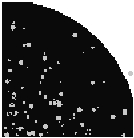
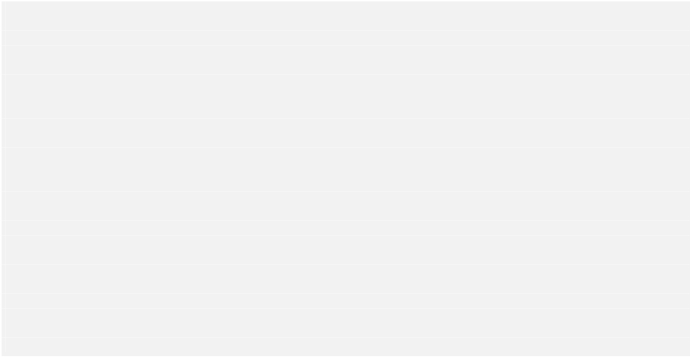
SELECT 3 AS bnum,

ST\_Buffer(ST\_GeomFromText(’LINESTRING(50 50,150 150,150 50)’), 10,’join= ←

bevel’) As geom

UNION ALL

SELECT 1 As bnum,



PostGIS 2.2.0dev 开发手册

486 / 686

ST\_Buffer(ST\_GeomFromText(’LINESTRING(60 50,150 150,150 50)’), 5,’join= ←

bevel’) As geom

),

-- define our canvas to be 1 to 1 pixel to geometry

canvas

AS (SELECT ST\_AddBand(ST\_MakeEmptyRaster(200,

200,

ST\_XMin(e)::integer, ST\_YMax(e)::integer, 1, -1, 0, 0) , ’8BUI’::text,0) As rast

FROM (SELECT ST\_Extent(geom) As e,

Max(ST\_SRID(geom)) As srid

from mygeoms

) As foo

),

rbands AS (SELECT ARRAY(SELECT ST\_MapAlgebraExpr(canvas.rast, ST\_AsRaster(m.geom, canvas ←

.rast, ’8BUI’, 100),

’[rast2.val]’, ’8BUI’, ’FIRST’, ’[rast2.val]’, ’[rast1.val]’) As rast

FROM mygeoms AS m CROSS JOIN canvas

ORDER BY m.bnum) As rasts

)

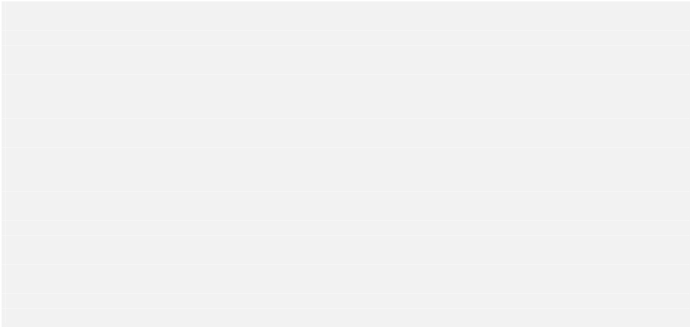
SELECT rasts[1] As rast1 , rasts[2] As rast2, rasts[3] As rast3, ST\_AddBand(

ST\_AddBand(rasts[1],rasts[2]), rasts[3]) As final\_rast

FROM rbands;

rast1

rast2



PostGIS 2.2.0dev 开发手册

487 / 686

rast3

ﬁnal\_rast

**样例: Overlay 2 meter boundary of select parcels over an aerial imagery**

-- Create new 3 band raster composed of first 2 clipped bands, and overlay of 3rd band with ←

our geometry

-- This query took 3.6 seconds on PostGIS windows 64-bit install

WITH pr AS

-- 注意 the order of operation: we clip all the rasters to dimensions of our region

(SELECT ST\_Clip(rast,ST\_Expand(geom,50) ) As rast, g.geom

FROM aerials.o\_2\_boston AS r INNER JOIN

-- union our parcels of interest so they form a single geometry we can later intersect with

(SELECT ST\_Union(ST\_Transform(the\_geom,26986)) AS geom

FROM landparcels WHERE pid IN(’0303890000’, ’0303900000’)) As g

ON ST\_Intersects(rast::geometry, ST\_Expand(g.geom,50))

),

-- we then union the raster shards together

-- ST\_Union on raster is kinda of slow but much faster the smaller you can get the rasters

-- therefore we want to clip first and then union

prunion AS

(SELECT ST\_AddBand(NULL, ARRAY[ST\_Union(rast,1),ST\_Union(rast,2),ST\_Union(rast,3)] ) As

clipped,geom

FROM pr

GROUP BY geom)

-- return our final raster which is the unioned shard with

-- with the overlay of our parcel boundaries

-- add first 2 bands, then mapalgebra of 3rd band + geometry

SELECT ST\_AddBand(ST\_Band(clipped,ARRAY[1,2])

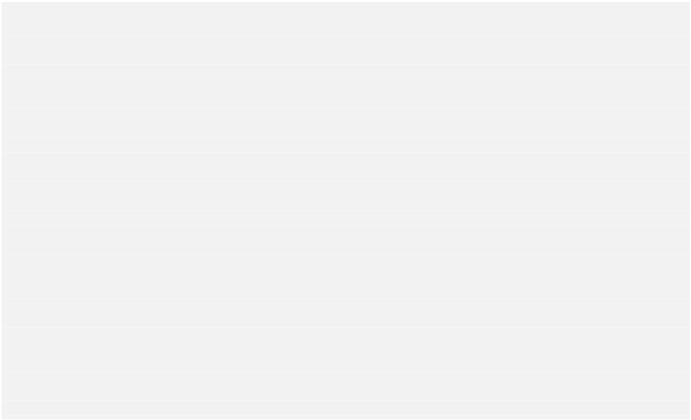
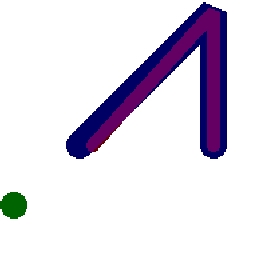
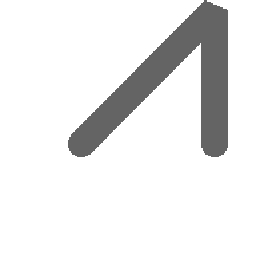
, ST\_MapAlgebraExpr(ST\_Band(clipped,3), ST\_AsRaster(ST\_Buffer(ST\_Boundary(geom),2), ←

clipped, ’8BUI’,250),

’[rast2.val]’, ’8BUI’, ’FIRST’, ’[rast2.val]’, ’[rast1.val]’) ) As rast

FROM prunion;

←



PostGIS 2.2.0dev 开发手册

488 / 686

The blue lines are the boundaries of select parcels

**相关请参考**

ST\_MapAlgebraExpr, ST\_AddBand, ST\_AsPNG, ST\_AsRaster, ST\_MapAlgebraFct, ST\_BandPixelType, ST\_GeoReference,

ST\_Value, ST\_Union, ST\_Union

**9.11.1.8**

**ST\_MapAlgebraFct**

ST\_MapAlgebraFct —输入参数是1个波段的函数版本。根据一个用户定义在输入栅格波段和pixeltype值的PostgreSQL函数，生成一个带有一个波段的栅格。如果没有指定波段是哪个，默认取波段1。

**用法**

raster ST\_MapAlgebraFct(raster rast, regprocedure onerasteruserfunc);

raster ST\_MapAlgebraFct(raster rast, regprocedure onerasteruserfunc, text[] VARIADIC args);

raster ST\_MapAlgebraFct(raster rast, text pixeltype, regprocedure onerasteruserfunc);

raster ST\_MapAlgebraFct(raster rast, text pixeltype, regprocedure onerasteruserfunc, text[] VARIADIC args);

raster ST\_MapAlgebraFct(raster rast, integer band, regprocedure onerasteruserfunc);

raster ST\_MapAlgebraFct(raster rast, integer band, regprocedure onerasteruserfunc, text[] VARIADIC args);

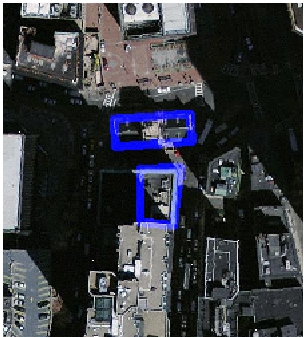
raster ST\_MapAlgebraFct(raster rast, integer band, text pixeltype, regprocedure onerasteruserfunc);

raster ST\_MapAlgebraFct(raster rast, integer band, text pixeltype, regprocedure onerasteruserfunc, text[] VARIADIC args);

**描述**

**警告**

ST\_MapAlgebraFct在2.1.0+版本不推荐使用，推荐使用函数ST\_MapAlgebra



PostGIS 2.2.0dev 开发手册

489 / 686

根据一个用户定义在输入栅格波段和pixeltype值的PostgreSQL函数（即参数onerasteruserfunc表示的函数名），生成一个带有一个波段的栅格。如果没有指定波段是哪个，默认取波段1。

新栅格将会和原始的输入栅格有相同的空间参考，宽度和高度，但是只会包含一个波段。

如果传递了参数pixeltype，那么新的栅格波段将会有一个波段的pixeltpye和传递参数值一样；

如果传递的为NULL，那么新栅格的波段的pixeltype和输入栅格波段一样。

参数onerasteruserfunc必须是SQL或PL/pgSQL的函数名，然后转换成regprocedure类型。

一个非常简单但也十分无用的PL/pgSQL 函数样例如下：

CREATE OR REPLACE FUNCTION simple\_function(pixel FLOAT, pos INTEGER[], VARIADIC args TEXT ←

[])

RETURNS FLOAT

AS $$ BEGIN

RETURN 0.0;

END; $$

LANGUAGE ’plpgsql’ IMMUTABLE;

参数userfunction对应的函数可能会接收2个或3个参数：一个float型值，一个可选的integer型数组，一个variadic text型数组。第一个参数是一个单独的栅格单元的值（不考虑栅格的数据类型）。第二个参数是当前处理单元的位置，格式是‘{x,y}’。第三个参数表示剩余的参数通过自定义函数userfunction传递给函数ST\_MapAlgebraFct的参数。

传递一个regprodedure 参数给一个SQL函数需要完整的函数名，然后转换成regprocedure类型。

要想传递上述样例PL/pgSQL 函数，那么SQL语句中的参数用法如下：

’simple\_function(float,integer[],text[])’::regprocedure

注意这个回调函数参数包含了回调函数的函数名，函数参数类型，包围函数名和参数类型的单引号以及进行regprocedure的转换。

userfunction第三个参数是一个variadic text型数组。所有的其他需要传递到ST\_MapAlgebraFct函数的自定义添加的参数都通过这个参数来传递给指定的userfunction

**注意**

关于关键字VARIADIC的信息，请参考PostgreSQL文档的章节"SQL

Functions with Variable Numbers of Arguments" section of [Query Language (SQL) Functions.](http://www.postgresql.org/docs/current/static/xfunc-sql.html)

**注意**

回调函数必须要传递参数text[], 无论你选择传递任意的一个参数给callbackfunction是否要进行处理。

可用版本：2.0.0

**样例**

Create a new 1 band raster from our original that is a function of modulo 2 of the original raster band.

ALTER TABLE dummy\_rast ADD COLUMN map\_rast raster;

CREATE FUNCTION mod\_fct(pixel float, pos integer[], variadic args text[])

RETURNS float

AS $$

BEGIN

RETURN pixel::integer % 2;

END;

$$



PostGIS 2.2.0dev 开发手册

490 / 686

LANGUAGE ’plpgsql’ IMMUTABLE;

UPDATE dummy\_rast SET map\_rast = ST\_MapAlgebraFct(rast,NULL,’mod\_fct(float,integer[],text ←

[])’::regprocedure) WHERE rid = 2;

SELECT ST\_Value(rast,1,i,j) As origval, ST\_Value(map\_rast, 1, i, j) As mapval

FROM dummy\_rast CROSS JOIN generate\_series(1, 3) AS i CROSS JOIN generate\_series(1,3) AS j

WHERE rid = 2;

origval | mapval

---------+--------

253 | 1

254 | 0

253 | 1

253 | 1

254 | 0

254 | 0

250 | 0

254 | 0

254 | 0

Create a new 1 band raster of pixel-type 2BUI from our original that is reclassiﬁed and set the nodata value to a passed parameter

to the user function (0).

ALTER TABLE dummy\_rast ADD COLUMN map\_rast2 raster;

CREATE FUNCTION classify\_fct(pixel float, pos integer[], variadic args text[])

RETURNS float

AS

$$

DECLARE

nodata float := 0;

BEGIN

IF NOT args[1] IS NULL THEN

nodata := args[1];

END IF;

IF pixel < 251 THEN

RETURN 1;

ELSIF pixel = 252 THEN

RETURN 2;

ELSIF pixel > 252 THEN

RETURN 3;

ELSE

RETURN nodata;

END IF;

END;

$$

LANGUAGE ’plpgsql’;

UPDATE dummy\_rast SET map\_rast2 = ST\_MapAlgebraFct(rast,’2BUI’,’classify\_fct(float,integer ←

[],text[])’::regprocedure, ’0’) WHERE rid = 2;

SELECT DISTINCT ST\_Value(rast,1,i,j) As origval, ST\_Value(map\_rast2, 1, i, j) As mapval

FROM dummy\_rast CROSS JOIN generate\_series(1, 5) AS i CROSS JOIN generate\_series(1,5) AS j

WHERE rid = 2;

origval | mapval

---------+--------

249 | 1

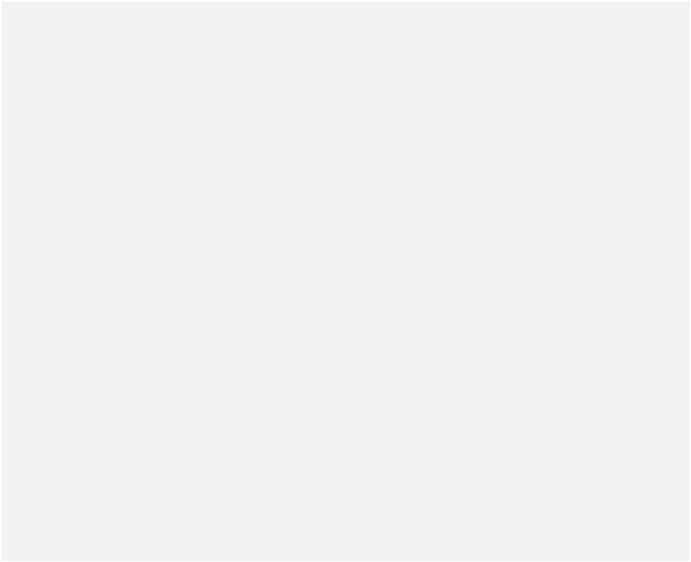
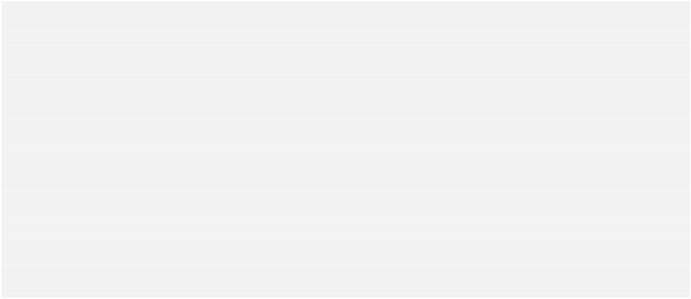
250 | 1

251 |

252 | 2

253 | 3

254 | 3



PostGIS 2.2.0dev 开发手册

491 / 686

SELECT ST\_BandPixelType(map\_rast2) As b1pixtyp

FROM dummy\_rast WHERE rid = 2;

b1pixtyp

----------

2BUI

original (column rast-view)

rast\_view\_ma

Create a new 3 band raster same pixel type from our original 3 band raster with ﬁrst band altered by map algebra and remaining

2 bands unaltered.

CREATE FUNCTION rast\_plus\_tan(pixel float, pos integer[], variadic args text[])

RETURNS float

AS

$$

BEGIN

RETURN tan(pixel) \* pixel;

END;

$$

LANGUAGE ’plpgsql’;

SELECT ST\_AddBand(

ST\_AddBand(

ST\_AddBand(

ST\_MakeEmptyRaster(rast\_view),

ST\_MapAlgebraFct(rast\_view,1,NULL,’rast\_plus\_tan(float,integer[],text[])’:: ←

regprocedure)

),

ST\_Band(rast\_view,2)

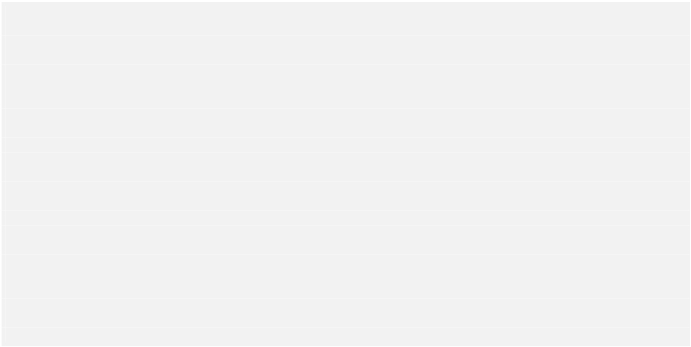
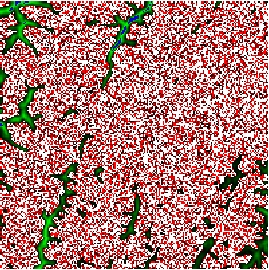
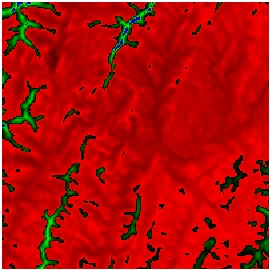
),

ST\_Band(rast\_view, 3) As rast\_view\_ma

)

FROM wind

WHERE rid=167;



PostGIS 2.2.0dev 开发手册

492 / 686

**相关请参考**

ST\_MapAlgebraExpr, ST\_BandPixelType, ST\_GeoReference, ST\_SetValue

**9.11.1.9**

**ST\_MapAlgebraFct**

ST\_MapAlgebraFct —输入参数是两个栅格波段的版本。根据一个用户定义在两个输入参数栅格波段和pixeltype值的

PostgreSQL函数，生成一个带有一个波段的栅格。如果没有指定每一个栅格的波段是哪个，那么默认都取波段1。

参数"extenttype"如果没有指定值，默认值是: INTERSECTION。

**用法**

raster ST\_MapAlgebraFct(raster rast1, raster rast2, regprocedure tworastuserfunc, text pixeltype=same\_as\_rast1, text extent-

type=INTERSECTION, text[] VARIADIC userargs);

raster ST\_MapAlgebraFct(raster rast1, integer band1, raster rast2, integer band2, regprocedure tworastuserfunc, text pixel-

type=same\_as\_rast1, text extenttype=INTERSECTION, text[] VARIADIC userargs);

**描述**

**警告**

ST\_MapAlgebraFct在2.1.0+版本不推荐使用，推荐使用函数ST\_MapAlgebra

根据一个用户定义在两个输入参数栅格rast1和rast2的PostgreSQL函数tworastuserfunc处理结果，生成一个带有一个波段的栅格。如果没有指定每一个栅格的波段是哪个，那么默认都取波段1。新栅格将会和原始的输入栅格有相同的空间参考，宽度和高度，但是只会包含一个波段。

如果传递了参数pixeltype，那么新的栅格波段将会有一个波段的pixeltpye和传递参数值一样；

如果传递的为NULL，那么新栅格的波段的pixeltype和输入栅格波段一样。

参数onerasteruserfunc必须是SQL或PL/pgSQL的函数名，然后转换成regprocedure类型。

一个PL/pgSQL函数样例如下：

CREATE OR REPLACE FUNCTION simple\_function\_for\_two\_rasters(pixel1 FLOAT, pixel2 FLOAT, pos

←

INTEGER[], VARIADIC args TEXT[])

RETURNS FLOAT

AS $$ BEGIN

RETURN 0.0;

END; $$

LANGUAGE ’plpgsql’ IMMUTABLE;

参数tworastuserfunc对应的函数可能会接收3个或4个参数：一个double型值，一个double型值，一个可选的integer型数组，一个variadic text型数组。第一个参数是一个栅格rast1的单独栅格单元的值（不考虑栅格的数据类型）。第二个参数是一个栅格rast2的单独栅格单元的值。第二个参数是当前处理单元的位置，格式是‘{x,y}’。第四个参数表示剩余的参数通过自定义函数tworastuserfunc传递给函数 ST\_MapAlgebraFct

传递一个regprodedure 参数给一个SQL函数需要完整的函数名，然后转换成regprocedure类型。

要想传递上述样例PL/pgSQL 函数，那么SQL语句中的参数用法如下：

’simple\_function(double precision, double precision, integer[], text[])’::regprocedure

注意这个回调函数参数包含了回调函数的函数名，函数参数类型，包围函数名和参数类型的单引号以及进行regprocedure

的转换。

tworastuserfunc的第三个参数是一个variadic text型数组。所有的其他需要传递到ST\_MapAlgebraFct函数的自定义添加的参数都通过这个参数来传递给指定的函数tworastuserfunc



PostGIS 2.2.0dev 开发手册

493 / 686

**注意**

关于关键字VARIADIC的信息，请参考PostgreSQL文档的章节"SQL

Functions with Variable Numbers of Arguments" section of [Query Language (SQL) Functions.](http://www.postgresql.org/docs/current/static/xfunc-sql.html)

**注意**

回调函数必须要传递参数text[], 无论你选择传递任意的一个参数给自定义的函数是否要进行处理。

可用版本：2.0.0

**样例: Overlaying rasters on a canvas as separate bands**

-- define our user defined function --

CREATE OR REPLACE FUNCTION raster\_mapalgebra\_union(

rast1 double precision,

rast2 double precision,

pos integer[],

VARIADIC userargs text[]

)

RETURNS double precision

AS $$

DECLARE

BEGIN

CASE

WHEN rast1 IS NOT NULL AND rast2 IS NOT NULL THEN

RETURN ((rast1 + rast2)/2.);

WHEN rast1 IS NULL AND rast2 IS NULL THEN

RETURN NULL;

WHEN rast1 IS NULL THEN

RETURN rast2;

ELSE

RETURN rast1;

END CASE;

RETURN NULL;

END;

$$ LANGUAGE ’plpgsql’ IMMUTABLE COST 1000;

-- prep our test table of rasters

DROP TABLE IF EXISTS map\_shapes;

CREATE TABLE map\_shapes(rid serial PRIMARY KEY, rast raster, bnum integer, descrip text);

INSERT INTO map\_shapes(rast,bnum, descrip)

WITH mygeoms

AS ( SELECT 2 As bnum, ST\_Buffer(ST\_Point(90,90),30) As geom, ’circle’ As descrip

UNION ALL

SELECT 3 AS bnum,

ST\_Buffer(ST\_GeomFromText(’LINESTRING(50 50,150 150,150 50)’), 15) As geom, ←

’big road’ As descrip

UNION ALL

SELECT 1 As bnum,

ST\_Translate(ST\_Buffer(ST\_GeomFromText(’LINESTRING(60 50,150 150,150 50)’), ←

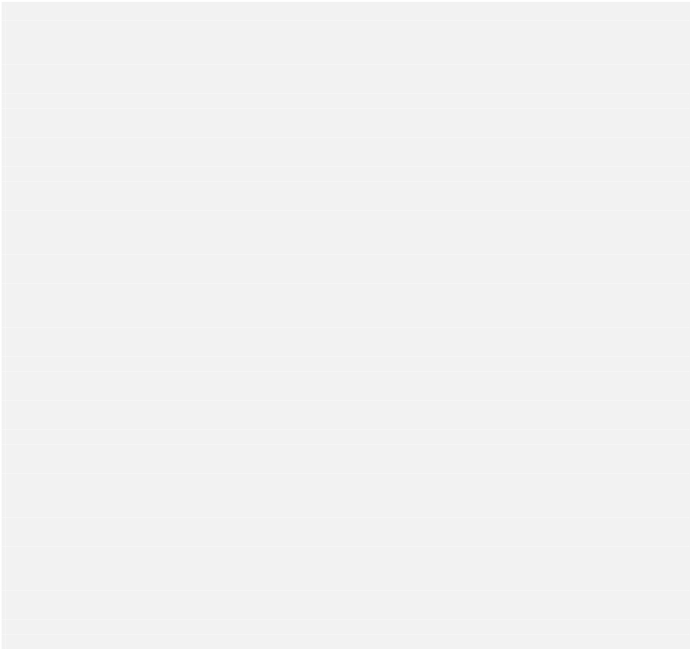
8,’join=bevel’), 10,-6) As geom, ’small road’ As descrip

),

-- define our canvas to be 1 to 1 pixel to geometry

canvas

AS ( SELECT ST\_AddBand(ST\_MakeEmptyRaster(250,



PostGIS 2.2.0dev 开发手册

494 / 686

250,

ST\_XMin(e)::integer, ST\_YMax(e)::integer, 1, -1, 0, 0 ) , ’8BUI’::text,0) As rast

FROM (SELECT ST\_Extent(geom) As e,

Max(ST\_SRID(geom)) As srid

from mygeoms

) As foo

)

-- return our rasters aligned with our canvas

SELECT ST\_AsRaster(m.geom, canvas.rast, ’8BUI’, 240) As rast, bnum, descrip

FROM mygeoms AS m CROSS JOIN canvas

UNION ALL

SELECT canvas.rast, 4, ’canvas’

FROM canvas;

-- Map algebra on single band rasters and then collect with ST\_AddBand

INSERT INTO map\_shapes(rast,bnum,descrip)

SELECT ST\_AddBand(ST\_AddBand(rasts[1], rasts[2]),rasts[3]), 4, ’map bands overlay fct union ←

(canvas)’

FROM (SELECT ARRAY(SELECT ST\_MapAlgebraFct(m1.rast, m2.rast,

’raster\_mapalgebra\_union(double precision, double precision, integer[], text[])’:: ←

regprocedure, ’8BUI’, ’FIRST’)

FROM map\_shapes As m1 CROSS JOIN map\_shapes As m2

WHERE m1.descrip = ’canvas’ AND m2.descrip <> ’canvas’ ORDER BY m2.bnum)

As rasts) As foo;

map bands overlay (canvas) (R: small road, G: circle, B: big road)

**User Deﬁned function that takes extra args**

CREATE OR REPLACE FUNCTION raster\_mapalgebra\_userargs(

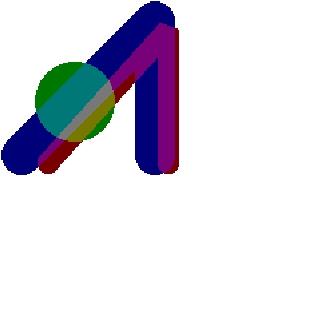
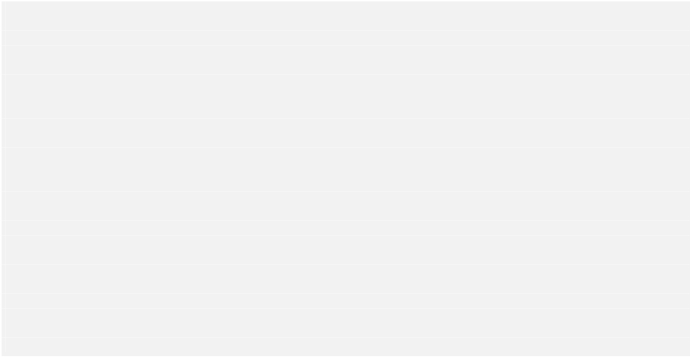
rast1 double precision,

rast2 double precision,

pos integer[],

VARIADIC userargs text[]

←



PostGIS 2.2.0dev 开发手册

495 / 686

)

RETURNS double precision

AS $$

DECLARE

BEGIN

CASE

WHEN rast1 IS NOT NULL AND rast2 IS NOT NULL THEN

RETURN least(userargs[1]::integer,(rast1 + rast2)/2.);

WHEN rast1 IS NULL AND rast2 IS NULL THEN

RETURN userargs[2]::integer;

WHEN rast1 IS NULL THEN

RETURN greatest(rast2,random()\*userargs[3]::integer)::integer;

ELSE

RETURN greatest(rast1, random()\*userargs[4]::integer)::integer;

END CASE;

RETURN NULL;

END;

$$ LANGUAGE ’plpgsql’ VOLATILE COST 1000;

SELECT ST\_MapAlgebraFct(m1.rast, 1, m1.rast, 3,

’raster\_mapalgebra\_userargs(double precision, double precision, integer[], text[])’:: ←

regprocedure,

’8BUI’, ’INTERSECT’, ’100’,’200’,’200’,’0’)

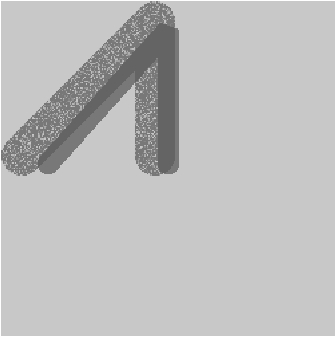
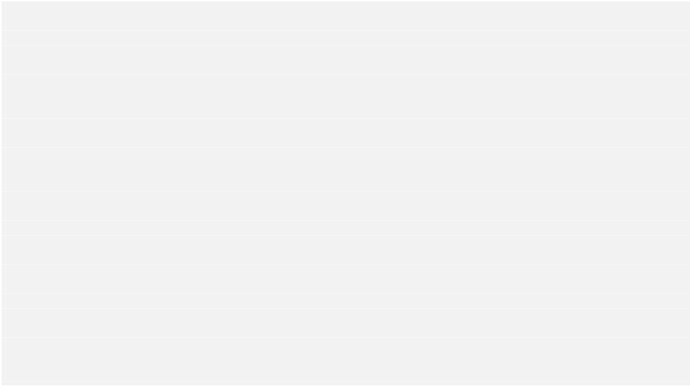
FROM map\_shapes As m1

WHERE m1.descrip = ’map bands overlay fct union (canvas)’;

user deﬁned with extra args and different bands from same raster

**相关请参考**

ST\_MapAlgebraExpr, ST\_BandPixelType, ST\_GeoReference, ST\_SetValue



PostGIS 2.2.0dev 开发手册

496 / 686

**9.11.1.10**

**ST\_MapAlgebraFctNgb**

ST\_MapAlgebraFctNgb —输入参数是一个波段的函数版本：该函数使用用户自定义的PostgreSQL 函数。该函数会返回一个栅格，其值是一个PLPGSQL自定义函数的结果，该自定义函数与输入栅格波段的近邻值有关。

**用法**

raster ST\_MapAlgebraFctNgb(raster rast, integer band, text pixeltype, integer ngbwidth, integer ngbheight, regprocedure on-

erastngbuserfunc, text nodatamode, text[] VARIADIC args);

**描述**

**警告**

ST\_MapAlgebraFctNgb在2.1.0+版本不推荐使用，推荐使用函数ST\_MapAlgebra

输入参数是一个波段的函数版本：该函数使用用户自定义的PostgreSQL 函数。该函数会返回一个栅格，其值是一个PLPGSQL自定义函数的结果，该自定义函数与输入栅格波段的近邻值有关。

用户自定义的函数会接收每个输入栅格波段的每个像素的相邻像素值作为一个数组，返回结果，然后再用返回结果去替换当前处理的像素的像素值。

**rast：**用户自定义的函数要处理的栅格。

**band**：要处理的栅格的波段号，默认是波段1。

**pixeltype**:输出栅格的pixeltype ，其值必须是ST\_BandPixelType输出值之一，可以省略或设置为NULL。如果没有传递或传递的是NULL，那么会被默认为是参数rast的pixel type类型。如果返回的结果比允许的pixel type值大，那么结果会被截断。

**ngbwidth**：相邻波段的宽度

**ngbheight**：相邻波段的高度。

**onerastngbuserfunc**：用户自定义的PLPGSQL/psql 函数，用于处理一个栅格的单个波段相邻的像素。第一个参数是一个2维数组，表示一个相邻的矩形像素。

**nodatamode**：定义当一个相邻像素是NODATA或NULL时，传递该参数的值给自定义的函数onerastngbuserfunc。

’ignore’: 遇到相邻任意NODATA值的像素都会忽略计算处理——这个标志必须传给回调函数，而用户自定义函数决定怎样忽略它。

’NULL’:遇到相邻任意NODATA值的像素计算都会引起返回的结果像素值为NULL——这种情况下，会跳过回调函数处理。

’value’:遇到相邻任意NODATA值的像素都会被参考像素（相邻栅格单元的中心像素）所替换。

注意如果该值为NODATA，那么其实现结果和取值为’NULL’一样（对于受影响的相邻栅格）

**args**:传递给自定义函数的参数。

可用版本：2.0.0

**样例**

Example utilize the katrina raster loaded as a single tile described in http://trac.osgeo.org/gdal/wiki/frmts\_wtkraster.html and

then prepared in the ST\_Rescale examples



PostGIS 2.2.0dev 开发手册

497 / 686

--

-- A simple ’callback’ user function that averages up all the values in a neighborhood.

--

CREATE OR REPLACE FUNCTION rast\_avg(matrix float[][], nodatamode text, variadic args text ←

[])

RETURNS float AS

$$

DECLARE

\_matrix float[][];

x1 integer;

x2 integer;

y1 integer;

y2 integer;

sum float;

BEGIN

\_matrix := matrix;

sum := 0;

FOR x in array\_lower(matrix, 1)..array\_upper(matrix, 1) LOOP

FOR y in array\_lower(matrix, 2)..array\_upper(matrix, 2) LOOP

sum := sum + \_matrix[x][y];

END LOOP;

END LOOP;

RETURN (sum\*1.0/(array\_upper(matrix,1)\*array\_upper(matrix,2) ))::integer ;

END;

$$

LANGUAGE ’plpgsql’ IMMUTABLE COST 1000;

-- now we apply to our raster averaging pixels within 2 pixels of each other in X and Y

direction --

SELECT ST\_MapAlgebraFctNgb(rast, 1, ’8BUI’, 4,4,

’rast\_avg(float[][], text, text[])’::regprocedure, ’NULL’, NULL) As nn\_with\_border

FROM katrinas\_rescaled

limit 1;

←

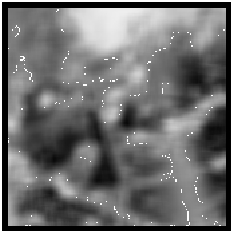
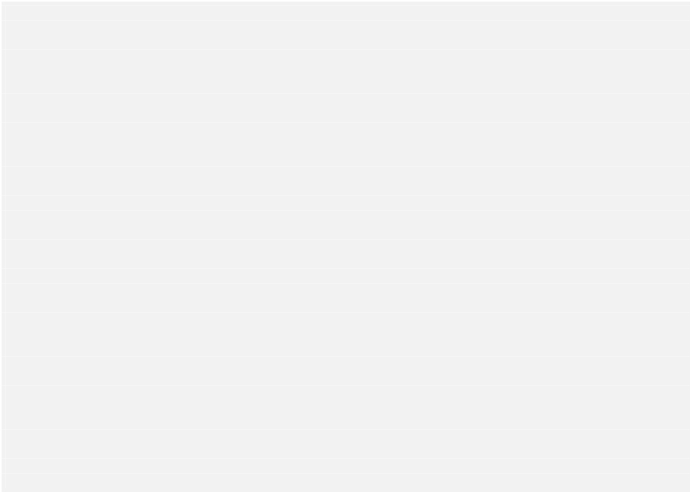
First band of our raster

**相关请参考**

ST\_MapAlgebraFct, ST\_MapAlgebraExpr, ST\_Rescale

new raster after averaging pixels withing 4x4 pixels of each

other



PostGIS 2.2.0dev 开发手册

498 / 686

**9.11.1.11**

**ST\_Reclass**

ST\_Reclass — 创建一个新的栅格，其波段类型将由将由原始的波段类型转换而来。如果没有指定nband值，那么默认取波段1。其他波段都不会改变。使用场景：把一个16BUI的波段转换成8BUI的波段，以便简化导出为可见的格式。

**用法**

raster ST\_Reclass(raster rast, integer nband, text reclassexpr, text pixeltype, double precision nodataval=NULL);

raster ST\_Reclass(raster rast, reclassarg[] VARIADIC reclassargset);

raster ST\_Reclass(raster rast, text reclassexpr, text pixeltype);

**描述**

创建一个新的栅格，其波段类型将由将由原始的波段类型转换而来。如果没有指定nband值，那么默认取波段1。其他波段都不会改变。使用场景：把一个16BUI的波段转换成8BUI的波段，以便简化导出为可见的格式。

根据一个用户定义在输入栅格波段的PostgreSQL代数表达式reclassexpr，执行生成一个带有一个波段的栅格。如果没有指定波段是哪个，默认取波段1。新栅格将会和原始的输入栅格有相同的空间参考，宽度和高度。没有指定的波段将不会修改类型。参考reclassarg获取一个有效的类型转换表述是怎样的。

新栅格的波段的像素类型被设定为pixeltype值。如果传递了参数reclassargset，那么每一个参数reclassarg会定义每一个生成波段的表现方式。

可用版本：2.0.0

**样例 Basic**

Create a new raster from the original where band 2 is converted from 8BUI to 4BUI and all values from 101-254 are set to nodata

value.

ALTER TABLE dummy\_rast ADD COLUMN reclass\_rast raster;

UPDATE dummy\_rast SET reclass\_rast = ST\_Reclass(rast,2,’0-87:1-10, 88-100:11-15,

101-254:0-0’, ’4BUI’,0) WHERE rid = 2;

←

SELECT i as col, j as row,

ST\_Value(reclass\_rast,

ST\_Value(reclass\_rast,

FROM dummy\_rast CROSS JOIN

ST\_Value(rast,2,i,j) As origval,

2, i, j) As reclassval,

2, i, j, false) As reclassval\_include\_nodata

generate\_series(1, 3) AS i CROSS JOIN generate\_series(1,3) AS j

WHERE rid = 2;

col | row | origval | reclassval | reclassval\_include\_nodata

-----+-----+---------+------------+---------------------------

1 | 1 | 78 | 9 | 9

2 | 1 | 98 | 14 | 14

3 | 1 | 122 | | 0

1 | 2 | 96 | 14 | 14

2 | 2 | 118 | | 0

3 | 2 | 180 | | 0

1 | 3 | 99 | 15 | 15

2 | 3 | 112 | | 0

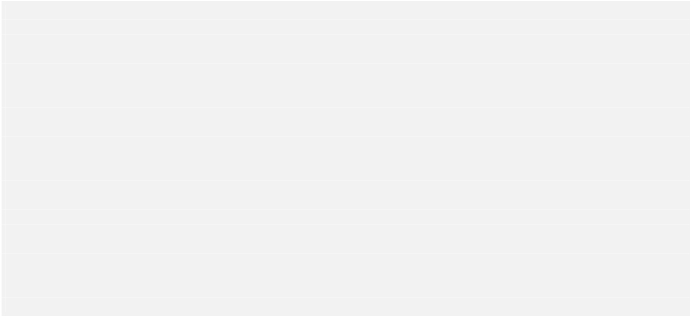
3 | 3 | 169 | | 0

**样例: Advanced using multiple reclassargs**

Create a new raster from the original where band 1,2,3 is converted to 1BB,4BUI, 4BUI respectively and reclassiﬁed. 注意 this

uses the variadic reclassarg argument which can take as input an indeﬁnite number of reclassargs (theoretically as many

bands as you have)



PostGIS 2.2.0dev 开发手册

499 / 686

UPDATE dummy\_rast SET reclass\_rast =

ST\_Reclass(rast,

ROW(2,’0-87]:1-10, (87-100]:11-15, (101-254]:0-0’, ’4BUI’,NULL)::reclassarg,

ROW(1,’0-253]:1, 254:0’, ’1BB’, NULL)::reclassarg,

ROW(3,’0-70]:1, (70-86:2, [86-150):3, [150-255:4’, ’4BUI’, NULL)::reclassarg

) WHERE rid = 2;

SELECT i as col, j as row,ST\_Value(rast,1,i,j) As ov1,

ST\_Value(reclass\_rast, 1, i, j) As

←

rv1,

ST\_Value(rast,2,i,j) As ov2, ST\_Value(reclass\_rast, 2, i, j) As rv2,

ST\_Value(rast,3,i,j) As ov3, ST\_Value(reclass\_rast, 3, i, j) As rv3

FROM dummy\_rast CROSS JOIN generate\_series(1, 3) AS i CROSS JOIN generate\_series(1,3) AS j

WHERE rid = 2;

col | row | ov1 | rv1 | ov2 | rv2 | ov3 | rv3

----+-----+-----+-----+-----+-----+-----+-----

1 | 1 | 253 | 1 | 78 | 9 | 70 | 1

2 | 1 | 254 | 0 | 98 | 14 | 86 | 3

3 | 1 | 253 | 1 | 122 | 0 | 100 | 3

1 | 2 | 253 | 1 | 96 | 14 | 80 | 2

2 | 2 | 254 | 0 | 118 | 0 | 108 | 3

3 | 2 | 254 | 0 | 180 | 0 | 162 | 4

1 | 3 | 250 | 1 | 99 | 15 | 90 | 3

2 | 3 | 254 | 0 | 112 | 0 | 108 | 3

3 | 3 | 254 | 0 | 169 | 0 | 175 | 4

**样例: Advanced Map a single band 32BF raster to multiple viewable bands**

Create a new 3 band (8BUI,8BUI,8BUI viewable raster) from a raster that has only one 32bf band

ALTER TABLE wind ADD COLUMN rast\_view raster;

UPDATE wind

set rast\_view = ST\_AddBand( NULL,

ARRAY[

ST\_Reclass(rast, 1,’0.1-10]:1-10,9-10]:11,(11-33:0’::text, ’8BUI’::text,0),

ST\_Reclass(rast,1, ’11-33):0-255,[0-32:0,(34-1000:0’::text, ’8BUI’::text,0),

ST\_Reclass(rast,1,’0-32]:0,(32-100:100-255’::text, ’8BUI’::text,0)

]

);

**相关请参考**

ST\_AddBand, ST\_Band, ST\_BandPixelType, ST\_MakeEmptyRaster, reclassarg, ST\_Value

**9.11.1.12**

**ST\_Union**

ST\_Union —取栅格瓦片的并集生成一个包含1个或更多波段的栅格。

**用法**

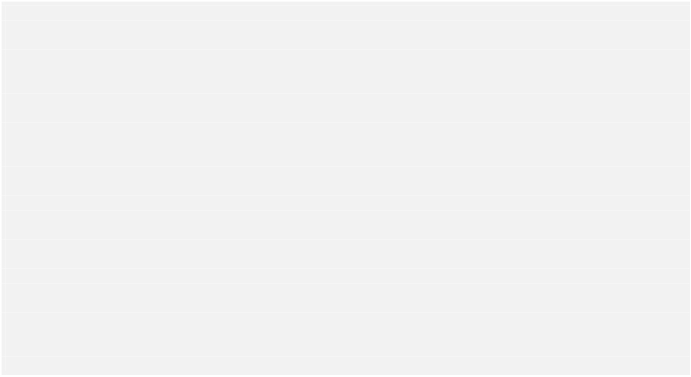
raster ST\_Union(setof raster rast);

raster ST\_Union(setof raster rast, unionarg[] unionargset);

raster ST\_Union(setof raster rast, integer nband);

raster ST\_Union(setof raster rast, text uniontype);

raster ST\_Union(setof raster rast, integer nband, text uniontype);



PostGIS 2.2.0dev 开发手册

500 / 686

**描述**

取栅格瓦片的并集生成一个包含1个或更多波段的栅格。返回栅格的边界是整个并集的边界。在交集的情况下，返回值的类型由参数uniontype来定义，uniontype值列表如下：LAST (default), FIRST, MIN, MAX, COUNT, SUM, MEAN, RANGE.

可用版本：2.0.0

版本提升： 2.1.0 提升了性能 (完全基于C实现了这个函数).

可用版本：2.1.0 引入了函数变体ST\_Union(rast, unionarg)

版本提升： 2.1.0修改了函数ST\_Union(rast) (variant 1)的实现，把所有输入的栅格的波段都union了起来，之前的版本只union了第一个波段。

版本提升： 2.1.0 修改了函数ST\_Union(rast, uniontype) (variant 4) 的实现，把所有输入的栅格的波段都union了起来，之前的版本只union了第一个波段。

**样例: Reconstitute a single band chunked raster tile**

-- this creates a single band from first band of raster tiles

-- that form the original file system tile

SELECT filename, ST\_Union(rast,1) As file\_rast

FROM sometable WHERE filename IN(’dem01’, ’dem02’) GROUP BY filename;

**样例: Return a multi-band raster that is the union of tiles intersecting geometry**

--

--

--

--

this creates a multi band raster collecting all the tiles that intersect a line

注意: In 2.0, this would have just returned a single band raster

, new union works on all bands by default

this is equivalent to unionarg: ARRAY[ROW(1, ’LAST’), ROW(2, ’LAST’), ROW(3, ’LAST’)]:: ←

unionarg[]

SELECT ST\_Union(rast)

FROM aerials.boston

WHERE ST\_Intersects(rast,

ST\_GeomFromText(’LINESTRING(230486 887771, 230500 88772)’,26986) ←

);

**样例: Return a multi-band raster that is the union of tiles intersecting geometry**

Here we use the longer syntax if we only wanted a subset of bands or we want to change order of bands

-- this creates a multi band raster collecting all the tiles that intersect a line

SELECT ST\_Union(rast,ARRAY[ROW(2, ’LAST’), ROW(1, ’LAST’), ROW(3, ’LAST’)]::unionarg[])

FROM aerials.boston

WHERE ST\_Intersects(rast,

ST\_GeomFromText(’LINESTRING(230486 887771, 230500 88772)’,26986) ←

);

**相关请参考**

unionarg, ST\_Envelope, ST\_ConvexHull, ST\_Clip, ST\_Union

**9.11.2**

**9.11.2.1**

**内置的地图代数回调函数**

**ST\_Distinct4ma**

ST\_Distinct4ma —栅格处理函数，计算近邻栅格单元唯一的像素值的个数。

PostGIS 2.2.0dev 开发手册

501 / 686

**用法**

ﬂoat8 ST\_Distinct4ma(ﬂoat8[][] matrix, text nodatamode, text[] VARIADIC args);

double precision ST\_Distinct4ma(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

**描述**

计算近邻栅格单元唯一的像素值的个数。

**注意**

函数变体1是一个专门的回调函数，用于当做函数ST\_MapAlgebraFctNgb的回调函数参数。

**注意**

函数变体2是一个专门的回调函数，用于当做函数ST\_MapAlgebra的回调函数参数。

**警告**

函数变体1不鼓励使用，因为函数ST\_MapAlgebraFctNgb在PostGIS 2.1.0版本已经不推荐使用了。

可用版本：2.0.0

版本提升： 2.1.0 添加函数变体2

**样例**

SELECT

rid,

st\_value(

st\_mapalgebrafctngb(rast, 1, NULL, 1, 1, ’st\_distinct4ma(float[][],text,text[])’:: ←

regprocedure, ’ignore’, NULL), 2, 2

)

FROM dummy\_rast

WHERE rid = 2;

rid | st\_value

-----+----------

2 | 3

(1 row)

**相关请参考**

ST\_MapAlgebraFctNgb, ST\_MapAlgebra, ST\_Min4ma, ST\_Max4ma, ST\_Sum4ma, ST\_Mean4ma, ST\_Distinct4ma, ST\_StdDev4ma

**9.11.2.2**

**ST\_InvDistWeight4ma**

ST\_InvDistWeight4ma — 栅格处理函数，它从像素的邻域计算一个像素插值。

**用法**

double precision ST\_InvDistWeight4ma(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);



PostGIS 2.2.0dev 开发手册

502 / 686

**描述**

使用Inverse Distance Weighted 方法计算一个像素的插值。

有两个可选参数可以用个参数userargs进行传递。第一个参数是power factor（就是下面等式的变量k），范围在0到1内，这个参数在Inverse Distance Weighted方程中用到。如果没有指定，默认值取1.第二个参数是权重比例weight percentage，这个参数只有感兴趣的像素的值在邻域的插值范围内才会起作用。如果没有指定并且感兴趣的像素有一个值，那么返回这个参数的值。基本的Inverse Distance Weighted方法等式如下:

k = power factor, a real number between 0 and 1

**注意**

这个函数是函数ST\_MapAlgebra的专用回调函数.

可用版本：2.1.0

**样例**

-- 需要样例

**相关请参考**

ST\_MapAlgebra, ST\_MinDist4ma

**9.11.2.3**

**ST\_Max4ma**

ST\_Max4ma — 栅格处理函数，它计算一个邻域的最大像素值。

**用法**

ﬂoat8 ST\_Max4ma(ﬂoat8[][] matrix, text nodatamode, text[] VARIADIC args);

double precision ST\_Max4ma(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

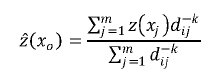
**描述**

栅格处理函数，它计算一个邻域的最大像素值。

对于函数变体2，可以在userargs参数中传递一个替代值给哪些值为NODATA 的像素。

**注意**

函数变体1是函数ST\_MapAlgebraFctNgb专用的回调函数



PostGIS 2.2.0dev 开发手册

503 / 686

**注意**

函数变体2是函数ST\_MapAlgebra专用的回调函数

**警告**

函数变体1不推荐使用，因为函数ST\_MapAlgebraFctNgb在PostGIS 2.1.0中已经不推荐使用了。

可用版本：2.0.0

版本提升： 2.1.0 添加函数变体2

**样例**

SELECT

rid,

st\_value(

st\_mapalgebrafctngb(rast, 1, NULL, 1, 1, ’st\_max4ma(float[][],text,text[])’:: ←

regprocedure, ’ignore’, NULL), 2, 2

)

FROM dummy\_rast

WHERE rid = 2;

rid | st\_value

-----+----------

2 | 254

(1 row)

**相关请参考**

ST\_MapAlgebraFctNgb, ST\_MapAlgebra, ST\_Min4ma, ST\_Sum4ma, ST\_Mean4ma, ST\_Range4ma, ST\_Distinct4ma, ST\_StdDev4ma

**9.11.2.4**

**ST\_Mean4ma**

ST\_Mean4ma —栅格处理函数，它计算一个邻域的平均像素值。

**用法**

ﬂoat8 ST\_Mean4ma(ﬂoat8[][] matrix, text nodatamode, text[] VARIADIC args);

double precision ST\_Mean4ma(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

**描述**

栅格处理函数，它计算一个邻域的平均像素值。

对于函数变体2，可以在userargs参数中传递一个替代值给哪些值为NODATA 的像素。

**注意**

函数变体1是函数ST\_MapAlgebraFctNgb专用的回调函数。



PostGIS 2.2.0dev 开发手册

504 / 686

**注意**

函数变体2是函数ST\_MapAlgebra专用的回调函数。

**警告**

函数变体1不推荐使用，因为函数ST\_MapAlgebraFctNgb在PostGIS 2.1.0中已经不推荐使用了。

可用版本：2.0.0

版本提升： 2.1.0添加函数变体2

**样例: Variant 1**

SELECT

rid,

st\_value(

st\_mapalgebrafctngb(rast, 1, ’32BF’, 1, 1, ’st\_mean4ma(float[][],text,text[])’:: ←

regprocedure, ’ignore’, NULL), 2, 2

)

FROM dummy\_rast

WHERE rid = 2;

rid |

st\_value

-----+------------------

2 | 253.222229003906

(1 row)

**样例: Variant 2**

SELECT

rid,

st\_value(

ST\_MapAlgebra(rast, 1, ’st\_mean4ma(double precision[][][], integer[][], text ←

[])’::regprocedure,’32BF’, ’FIRST’, NULL, 1, 1)

, 2, 2)

FROM dummy\_rast

WHERE rid = 2;

rid |

st\_value

-----+------------------

2 | 253.222229003906

(1 row)

**相关请参考**

ST\_MapAlgebraFctNgb, ST\_MapAlgebra, ST\_Min4ma, ST\_Max4ma, ST\_Sum4ma, ST\_Range4ma, ST\_StdDev4ma

**9.11.2.5**

**ST\_Min4ma**

ST\_Min4ma —栅格处理函数，它计算一个邻域的最小像素值。



PostGIS 2.2.0dev 开发手册

505 / 686

**用法**

ﬂoat8 ST\_Min4ma(ﬂoat8[][] matrix, text nodatamode, text[] VARIADIC args);

double precision ST\_Min4ma(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

**描述**

栅格处理函数，它计算一个邻域的最小像素值。

对于函数变体2，可以在userargs参数中传递一个替代值给哪些值为NODATA 的像素。

**注意**

函数变体1是函数ST\_MapAlgebraFctNgb专用的回调函数。

**注意**

函数变体2是函数ST\_MapAlgebra专用的回调函数。

**警告**

函数变体1不推荐使用，因为函数ST\_MapAlgebraFctNgb在PostGIS 2.1.0中已经不推荐使用了。

可用版本：2.0.0

版本提升： 2.1.0添加函数变体2

**样例**

SELECT

rid,

st\_value(

st\_mapalgebrafctngb(rast, 1, NULL, 1, 1, ’st\_min4ma(float[][],text,text[])’:: ←

regprocedure, ’ignore’, NULL), 2, 2

)

FROM dummy\_rast

WHERE rid = 2;

rid | st\_value

-----+----------

2 | 250

(1 row)

**相关请参考**

ST\_MapAlgebraFctNgb, ST\_MapAlgebra, ST\_Max4ma, ST\_Sum4ma, ST\_Mean4ma, ST\_Range4ma, ST\_Distinct4ma, ST\_StdDev4m

**9.11.2.6**

**ST\_MinDist4ma**

ST\_MinDist4ma —栅格处理函数，返回感兴趣的像素和一个邻域像素之间的最小距离（距离单位是像素的个数）



PostGIS 2.2.0dev 开发手册

506 / 686

**用法**

double precision ST\_MinDist4ma(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

**描述**

栅格处理函数，返回感兴趣的像素和一个邻域像素之间的最小距离（距离单位是像素的个数）

**注意**

这个函数用途是提供提供点的数据信息用于推断从函数ST\_InvDistWeight4ma返回的感兴趣的像素的插值。这个函数在邻域特别稀疏时候很有用。

**注意**

这个函数是函数ST\_MapAlgebra专用的回调函数。

可用版本：2.1.0

**样例**

-- 需要样例（需要样例，译者注：原文就没有提供）

**相关请参考**

ST\_MapAlgebra, ST\_InvDistWeight4ma

**9.11.2.7**

**ST\_Range4ma**

ST\_Range4ma — 栅格处理函数，计算邻域的像素值范围。

**用法**

ﬂoat8 ST\_Range4ma(ﬂoat8[][] matrix, text nodatamode, text[] VARIADIC args);

double precision ST\_Range4ma(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

**描述**

栅格处理函数，计算邻域的像素值范围。

对于函数变体2，可以在userargs参数中传递一个替代值给哪些值为NODATA 的像素。

**注意**

函数变体1是函数ST\_MapAlgebraFctNgb专用的回调函数。



PostGIS 2.2.0dev 开发手册

507 / 686

**注意**

函数变体2是函数ST\_MapAlgebra专用的回调函数。

**警告**

函数变体1不推荐使用，因为函数ST\_MapAlgebraFctNgb在PostGIS 2.1.0中已经不推荐使用了。

可用版本：2.0.0

版本提升： 2.1.0 添加函数变体2

**样例**

SELECT

rid,

st\_value(

st\_mapalgebrafctngb(rast, 1, NULL, 1, 1, ’st\_range4ma(float[][],text,text[])’:: ←

regprocedure, ’ignore’, NULL), 2, 2

)

FROM dummy\_rast

WHERE rid = 2;

rid | st\_value

-----+----------

2 | 4

(1 row)

**相关请参考**

ST\_MapAlgebraFctNgb, ST\_MapAlgebra, ST\_Min4ma, ST\_Max4ma, ST\_Sum4ma, ST\_Mean4ma, ST\_Distinct4ma, ST\_StdDev4ma

**9.11.2.8**

**ST\_StdDev4ma**

ST\_StdDev4ma —栅格处理函数，计算邻域的像素的标准差。

**用法**

ﬂoat8 ST\_StdDev4ma(ﬂoat8[][] matrix, text nodatamode, text[] VARIADIC args);

double precision ST\_StdDev4ma(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

**描述**

栅格处理函数，计算邻域的像素的标准差。

**注意**

函数变体1是函数ST\_MapAlgebraFctNgb专用的回调函数。



PostGIS 2.2.0dev 开发手册

508 / 686

**注意**

函数变体2是函数ST\_MapAlgebra专用的回调函数。

**警告**

函数变体1不推荐使用，因为函数ST\_MapAlgebraFctNgb在PostGIS 2.1.0中已经不推荐使用了。

可用版本：2.0.0

版本提升： 2.1.0 添加函数变体2

**样例**

SELECT

rid,

st\_value(

st\_mapalgebrafctngb(rast, 1, ’32BF’, 1, 1, ’st\_stddev4ma(float[][],text,text[])’:: ←

regprocedure, ’ignore’, NULL), 2, 2

)

FROM dummy\_rast

WHERE rid = 2;

rid |

st\_value

-----+------------------

2 | 1.30170822143555

(1 row)

**相关请参考**

ST\_MapAlgebraFctNgb, ST\_MapAlgebra, ST\_Min4ma, ST\_Max4ma, ST\_Sum4ma, ST\_Mean4ma, ST\_Distinct4ma, ST\_StdDev4ma

**9.11.2.9**

**ST\_Sum4ma**

ST\_Sum4ma — 栅格处理函数，计算一个邻域里面像素值的和。

**用法**

ﬂoat8 ST\_Sum4ma(ﬂoat8[][] matrix, text nodatamode, text[] VARIADIC args);

double precision ST\_Sum4ma(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

**描述**

栅格处理函数，计算一个邻域里面像素值的和。

对于函数变体2，可以在userargs参数中传递一个替代值给哪些值为NODATA 的像素。

**注意**

函数变体1是函数ST\_MapAlgebraFctNgb专用的回调函数。



PostGIS 2.2.0dev 开发手册

509 / 686

**注意**

函数变体2是函数ST\_MapAlgebra专用的回调函数。

**警告**

函数变体1不推荐使用，因为函数ST\_MapAlgebraFctNgb在PostGIS 2.1.0中已经不推荐使用了。

可用版本：2.0.0

版本提升： 2.1.0 添加函数变体2

**样例**

SELECT

rid,

st\_value(

st\_mapalgebrafctngb(rast, 1, ’32BF’, 1, 1, ’st\_sum4ma(float[][],text,text[])’:: ←

regprocedure, ’ignore’, NULL), 2, 2

)

FROM dummy\_rast

WHERE rid = 2;

rid | st\_value

-----+----------

2 | 2279

(1 row)

**相关请参考**

ST\_MapAlgebraFctNgb, ST\_MapAlgebra, ST\_Min4ma, ST\_Max4ma, ST\_Mean4ma, ST\_Range4ma, ST\_Distinct4ma, ST\_StdDev4ma

**9.11.3**

**9.11.3.1**

**数字高程模型——DEM (Elevation)**

**ST\_Aspect**

ST\_Aspect —获得栅格表面的坡向 (默认单位是度)。对于地形分析很有用。

**用法**

raster ST\_Aspect(raster rast, integer band=1, text pixeltype=32BF, text units=DEGREES, boolean interpolate\_nodata=FALSE);

raster ST\_Aspect(raster rast, integer band, raster customextent, text pixeltype=32BF, text units=DEGREES, boolean interpo-

late\_nodata=FALSE);

**描述**

获得栅格表面的坡向 (默认单位是度)。利用地图代数并把坡向用于邻域像素。

参数units是坡向的单位。坡向单位可能的值是：RADIANS, DEGREES (默认单位)。

当单位参数units = RADIANS时，取值范围是0 到 2 \* pi弧度之间，从正北方向顺时针开始计算。

当单位参数units = DEGREES时，取值范围在0到360度之间，从正北方向顺时针测量。

如果像素的坡度为0，那么像素的坡向是-1。



PostGIS 2.2.0dev 开发手册

510 / 686

**注意**

更多关于Slope（坡度）, Aspect（坡向）和 Hillshade（山体阴影）的信息请参考ESRI - How hillshade works和 [ERDAS Field](http://geospatial.intergraph.com/fieldguide/wwhelp/wwhimpl/common/html/wwhelp.htm?context=FieldGuide&file=Aspect_Images.html)

[Guide - Aspect Images.](http://geospatial.intergraph.com/fieldguide/wwhelp/wwhimpl/common/html/wwhelp.htm?context=FieldGuide&file=Aspect_Images.html)

可用版本：2.0.0

版本提升： 2.1.0 使用函数ST\_MapAlgebra()计算，添加函数可选参数interpolate\_nodata

版本变更：2.1.0 在早先的版本，返回的值是以弧度为单位的，现在返回的值默认是以度为单位的。

**样例: Variant 1**

WITH foo AS (

SELECT ST\_SetValues(

ST\_AddBand(ST\_MakeEmptyRaster(5, 5, 0, 0, 1, -1, 0, 0, 0), 1, ’32BF’, 0, -9999),

1, 1, 1, ARRAY[

[1, 1, 1, 1, 1],

[1, 2, 2, 2, 1],

[1, 2, 3, 2, 1],

[1, 2, 2, 2, 1],

[1, 1, 1, 1, 1]

]::double precision[][]

) AS rast

)

SELECT

ST\_DumpValues(ST\_Aspect(rast, 1, ’32BF’))

FROM foo

s

-----------------------------------------------------------------------------------------------------

----------------------------------

(1,"{{315,341.565063476562,0,18.4349479675293,45},{288.434936523438,315,0,45,71.5650482177734},{270,

2227,180,161.565048217773,135}}")

(1 row)

**样例: Variant 2**

Complete example of tiles of a coverage. This query only works with PostgreSQL 9.1 or higher.

WITH foo AS (

SELECT ST\_Tile(

ST\_SetValues(

ST\_AddBand(

ST\_MakeEmptyRaster(6, 6, 0, 0, 1, -1, 0, 0, 0),

1, ’32BF’, 0, -9999

),

1, 1, 1, ARRAY[

[1, 1, 1, 1, 1, 1],

[1, 1, 1, 1, 2, 1],

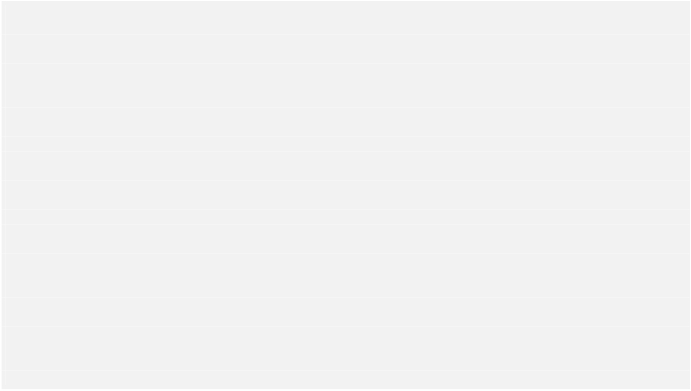
[1, 2, 2, 3, 3, 1],

[1, 1, 3, 2, 1, 1],

[1, 2, 2, 1, 2, 1],

[1, 1, 1, 1, 1, 1]

]::double precision[]



PostGIS 2.2.0dev 开发手册

511 / 686

),

2, 2

) AS rast

)

SELECT

t1.rast,

ST\_Aspect(ST\_Union(t2.rast), 1, t1.rast)

FROM foo t1

CROSS JOIN foo t2

WHERE ST\_Intersects(t1.rast, t2.rast)

GROUP BY t1.rast;

**相关请参考**

ST\_MapAlgebra, ST\_TRI, ST\_TPI, ST\_Roughness, ST\_HillShade, ST\_Slope

**9.11.3.2**

**ST\_HillShade**

ST\_HillShade — 根据输入参数azimuth, altitude, brightness和 scale，返回栅格波段的山体阴影。

**用法**

raster ST\_HillShade(raster rast, integer band=1, text pixeltype=32BF, double precision azimuth=315, double precision alti-

tude=45, double precision max\_bright=255, double precision scale=1.0, boolean interpolate\_nodata=FALSE);

raster ST\_HillShade(raster rast, integer band, raster customextent, text pixeltype=32BF, double precision azimuth=315, double

precision altitude=45, double precision max\_bright=255, double precision scale=1.0, boolean interpolate\_nodata=FALSE);

**描述**

根据输入参数azimuth, altitude, brightness和 scale，返回栅格波段的山体阴影。使用地图代数并把山体阴影高程用于邻域像素。返回像素值在0到255之间。

azimuth：是一个值在0到360度之间按照正北方向顺时针测量的值。

altitude ：取值在0到90度之间，0度表示处于地平线，而90度表示垂直。

max\_bright：取值在0到255之间，0表示没有亮度，而255是最大亮度。

scale：是垂直单位与水平单位的比值。例如Feet:LatLon ，使用scale=370400,对于 Meters:LatLon 使用scale=111120.

如果参数interpolate\_nodata = TRUE, 输入栅格的NODATA 像素值将在计算山体阴影之前使用函数ST\_InvDistWeight4ma进行插值。

**注意**

更多关于Hillshade（山体阴影）信息，请参考[How hillshade works.](http://webhelp.esri.com/arcgisdesktop/9.3/index.cfm?TopicName=How%20Hillshade%20works)

可用版本：2.0.0

版本提升： 2.1.0 利用函数 ST\_MapAlgebra() 计算，并且添加了可选参数interpolate\_nodata

版本变更：2.1.0 在早前的版本中，参数azimuth 和 altitude使用弧度表示，现在azimuth 和 altitude 都使用度来表示。



PostGIS 2.2.0dev 开发手册

512 / 686

**样例: Variant 1**

WITH foo AS (

SELECT ST\_SetValues(

ST\_AddBand(ST\_MakeEmptyRaster(5, 5, 0, 0, 1, -1, 0, 0, 0), 1, ’32BF’, 0, -9999),

1, 1, 1, ARRAY[

[1, 1, 1, 1, 1],

[1, 2, 2, 2, 1],

[1, 2, 3, 2, 1],

[1, 2, 2, 2, 1],

[1, 1, 1, 1, 1]

]::double precision[][]

) AS rast

)

SELECT

ST\_DumpValues(ST\_Hillshade(rast, 1, ’32BF’))

FROM foo

-----------------------------------------------------------------------------------------------------

-----------------------------------------------------------------------

(1,"{{NULL,NULL,NULL,NULL,NULL},{NULL,251.32763671875,220.749786376953,147.224319458008, ←

NULL},{NULL,220.749786376953,180.312225341797,67.7497863769531,NULL},{NULL ←

,147.224319458008

,67.7497863769531,43.1210060119629,NULL},{NULL,NULL,NULL,NULL,NULL}}")

(1 row)

**样例: Variant 2**

Complete example of tiles of a coverage. This query only works with PostgreSQL 9.1 or higher.

WITH foo AS (

SELECT ST\_Tile(

ST\_SetValues(

ST\_AddBand(

ST\_MakeEmptyRaster(6, 6, 0, 0, 1, -1, 0, 0, 0),

1, ’32BF’, 0, -9999

),

1, 1, 1, ARRAY[

[1, 1, 1, 1, 1, 1],

[1, 1, 1, 1, 2, 1],

[1, 2, 2, 3, 3, 1],

[1, 1, 3, 2, 1, 1],

[1, 2, 2, 1, 2, 1],

[1, 1, 1, 1, 1, 1]

]::double precision[]

),

2, 2

) AS rast

)

SELECT

t1.rast,

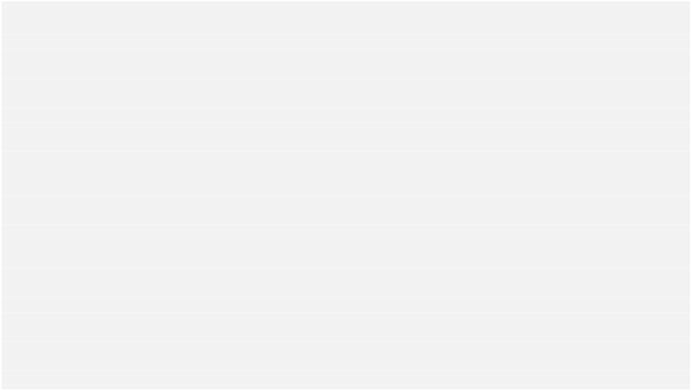
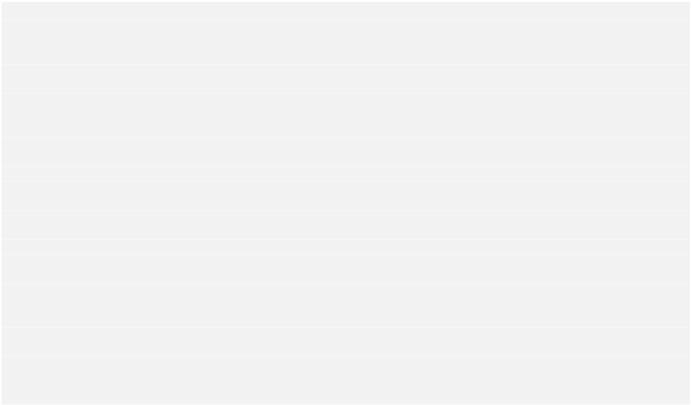
ST\_Hillshade(ST\_Union(t2.rast), 1, t1.rast)

FROM foo t1

CROSS JOIN foo t2

WHERE ST\_Intersects(t1.rast, t2.rast)

GROUP BY t1.rast;



PostGIS 2.2.0dev 开发手册

513 / 686

**相关请参考**

ST\_MapAlgebra, ST\_TRI, ST\_TPI, ST\_Roughness, ST\_Aspect, ST\_Slope

**9.11.3.3**

**ST\_Roughness**

ST\_Roughness —返回一个DEM模型的“粗糙程度”的栅格。

**用法**

raster ST\_Roughness(raster rast, integer nband, raster customextent, text pixeltype="32BF" , boolean interpolate\_nodata=FALSE

);

**描述**

通过计算给定区域最大值和最小值的差计算一个DEM模型的粗糙程度。

可用版本：2.1.0

**样例**

-- 需要样例

**相关请参考**

ST\_MapAlgebra, ST\_TRI, ST\_TPI, ST\_Slope, ST\_HillShade, ST\_Aspect

**9.11.3.4**

**ST\_Slope**

ST\_Slope —返回一个栅格波段的坡度（默认以度为单位）。对于地形分析很有用。

**用法**

raster ST\_Slope(raster rast, integer nband=1, text pixeltype=32BF, text units=DEGREES, double precision scale=1.0, boolean

interpolate\_nodata=FALSE);

raster ST\_Slope(raster rast, integer nband, raster customextent, text pixeltype=32BF, text units=DEGREES, double precision

scale=1.0, boolean interpolate\_nodata=FALSE);

**描述**

返回一个栅格波段的坡度（默认以度为单位）。将地图代数方法应用于邻域像素的坡度方程。

参数units表示坡度的单位。可能值有： RADIANS, DEGREES (默认取值), PERCENT。

scale：是垂直单位与水平单位的比值。例如Feet:LatLon ，使用scale=370400,对于 Meters:LatLon 使用scale=111120.

如果参数interpolate\_nodata = TRUE, 输入栅格的NODATA 像素值将在计算表面坡度之前使用函数ST\_InvDistWeight4ma进行插值。

**注意**

更多关于Slope（坡度）, Aspect（坡向）和 Hillshade（山体阴影），请参考ESRI - How hillshade works和 [ERDAS Field](http://geospatial.intergraph.com/fieldguide/wwhelp/wwhimpl/common/html/wwhelp.htm?context=FieldGuide&file=Slope_Images.html)

[Guide - Slope Images.](http://geospatial.intergraph.com/fieldguide/wwhelp/wwhimpl/common/html/wwhelp.htm?context=FieldGuide&file=Slope_Images.html)



PostGIS 2.2.0dev 开发手册

514 / 686

可用版本：2.0.0

版本提升： 2.1.0 使用函数ST\_MapAlgebra() 并添加可选参数units, scale, interpolate\_nodata

版本变更：2.1.0 在早期的版本，返回的值单位是radians（弧度），现在默认返回的是degrees（度）

**样例: Variant 1**

WITH foo AS (

SELECT ST\_SetValues(

ST\_AddBand(ST\_MakeEmptyRaster(5, 5, 0, 0, 1, -1, 0, 0, 0), 1, ’32BF’, 0, -9999),

1, 1, 1, ARRAY[

[1, 1, 1, 1, 1],

[1, 2, 2, 2, 1],

[1, 2, 3, 2, 1],

[1, 2, 2, 2, 1],

[1, 1, 1, 1, 1]

]::double precision[][]

) AS rast

)

SELECT

ST\_DumpValues(ST\_Slope(rast, 1, ’32BF’))

FROM foo

st\_dumpvalues

-----------------------------------------------------------------------------------------------------

-----------------------------------------------------------------------------------------------------

---------------------------------------------------------------------

(1,"{{10.0249881744385,21.5681285858154,26.5650520324707,21.5681285858154,10.0249881744385},{21.5681

{26.5650520324707,36.8698959350586,0,36.8698959350586,26.5650520324707},{21.5681285858154,35.26438903

5681285858154,26.5650520324707,21.5681285858154,10.0249881744385}}")

(1 row)

**样例: Variant 2**

Complete example of tiles of a coverage. This query only works with PostgreSQL 9.1 or higher.

WITH foo AS (

SELECT ST\_Tile(

ST\_SetValues(

ST\_AddBand(

ST\_MakeEmptyRaster(6, 6, 0, 0, 1, -1, 0, 0, 0),

1, ’32BF’, 0, -9999

),

1, 1, 1, ARRAY[

[1, 1, 1, 1, 1, 1],

[1, 1, 1, 1, 2, 1],

[1, 2, 2, 3, 3, 1],

[1, 1, 3, 2, 1, 1],

[1, 2, 2, 1, 2, 1],

[1, 1, 1, 1, 1, 1]

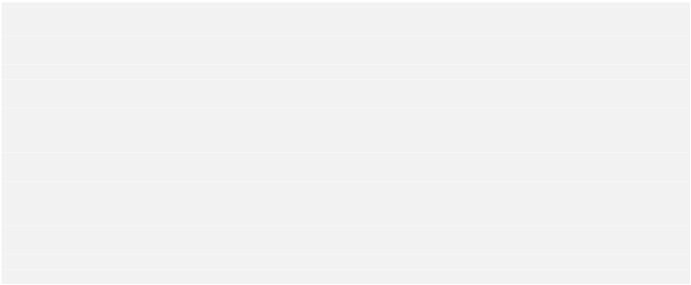
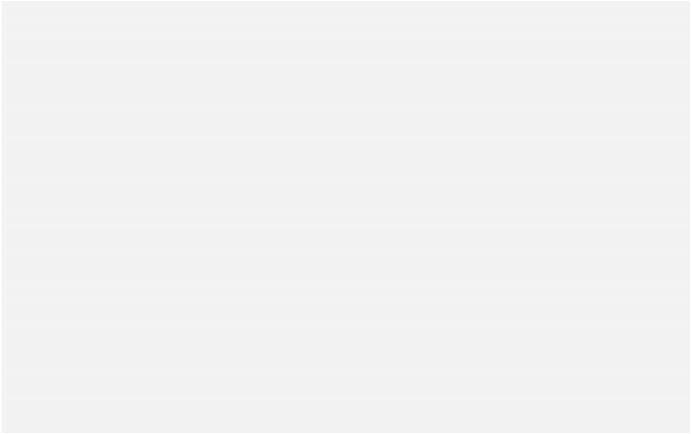
]::double precision[]

),

2, 2

) AS rast

)



PostGIS 2.2.0dev 开发手册

515 / 686

SELECT

t1.rast,

ST\_Slope(ST\_Union(t2.rast), 1, t1.rast)

FROM foo t1

CROSS JOIN foo t2

WHERE ST\_Intersects(t1.rast, t2.rast)

GROUP BY t1.rast;

**相关请参考**

ST\_MapAlgebra, ST\_TRI, ST\_TPI, ST\_Roughness, ST\_HillShade, ST\_Aspect

**9.11.3.5**

**ST\_TPI**

ST\_TPI —返回一个带有地形位置指数的栅格。

**用法**

raster ST\_TPI(raster rast, integer nband, raster customextent, text pixeltype="32BF" , boolean interpolate\_nodata=FALSE );

**描述**

返回一个带有地形位置指数的栅格，地形位置指数的定义是中心点高程值减去周围高程的平均值。

**注意**

这个函数只支持一个栅格平均计算。

可用版本：2.1.0

**样例**

-- 需要样例

**相关请参考**

ST\_MapAlgebra, ST\_TRI, ST\_Roughness, ST\_Slope, ST\_HillShade, ST\_Aspect

**9.11.3.6**

**ST\_TRI**

ST\_TRI — 返回一个带有地形耐用指数的栅格。

**用法**

raster ST\_TRI(raster rast, integer nband, raster customextent, text pixeltype="32BF" , boolean interpolate\_nodata=FALSE );



PostGIS 2.2.0dev 开发手册

516 / 686

**描述**

TRI是指中心点高程与周围高程的差的平均值。

**注意**

这个函数只支持一个栅格平均计算

可用版本：2.1.0

**样例**

-- 需要样例

**相关请参考**

ST\_MapAlgebra, ST\_Roughness, ST\_TPI, ST\_Slope, ST\_HillShade, ST\_Aspect

**9.11.4**

**9.11.4.1**

**栅格转换成几何对象函数**

**Box3D**

Box3D — 返回包围栅格“壳”的box 3d几何对象的形式。

**用法**

box3d Box3D(raster rast);

**描述**

返回包围栅格“壳”的box 3d几何对象的形式。

多边形由缓冲区((MINX, MINY), (MAXX, MAXY))的角点定义。

版本变更：2.0.0在2.0之前的版本，返回的是box2d几何对象，因为box2d是一个不推荐的类型，因此改成了返回box3d几何对象。

**样例**

SELECT

rid,

Box3D(rast) AS rastbox

FROM dummy\_rast;

rid |

rastbox

----+-------------------------------------------------

1 | BOX3D(0.5 0.5 0,20.5 60.5 0)

2 | BOX3D(3427927.75 5793243.5 0,3427928 5793244 0)



PostGIS 2.2.0dev 开发手册

517 / 686

**相关请参考**

ST\_Envelope

**9.11.4.2**

**ST\_ConvexHull**

ST\_ConvexHull — 返回一个栅格的凸包几何对象，Return the convex hull geometry of the raster including pixel values equal to BandNoDataValue. For regular

shaped and non-skewed rasters, this gives the same result as ST\_Envelope so only useful for irregularly shaped or skewed rasters.

**用法**

geometry ST\_ConvexHull(raster rast);

**描述**

Return the convex hull geometry of the raster including the NoDataBandValue band pixels. For regular shaped and non-skewed

rasters, this gives more or less the same result as ST\_Envelope so only useful for irregularly shaped or skewed rasters.

**注意**

ST\_Envelope ﬂoors the coordinates and hence add a little buffer around the raster so the answer is subtly different from

ST\_ConvexHull which does not ﬂoor.

**样例**

Refer to PostGIS Raster Speciﬁcation for a diagram of this.

-- 注意 envelope and convexhull are more or less the same

SELECT ST\_AsText(ST\_ConvexHull(rast)) As convhull,

ST\_AsText(ST\_Envelope(rast)) As env

FROM dummy\_rast WHERE rid=1;

convhull

|

env

--------------------------------------------------------+------------------------------------ ←

POLYGON((0.5 0.5,20.5 0.5,20.5 60.5,0.5 60.5,0.5 0.5)) | POLYGON((0 0,20 0,20 60,0 60,0 0) ←

)

-- now we skew the raster

-- note how the convex hull and envelope are now different

SELECT ST\_AsText(ST\_ConvexHull(rast)) As convhull,

ST\_AsText(ST\_Envelope(rast)) As env

FROM (SELECT ST\_SetRotation(rast, 0.1, 0.1) As rast

FROM dummy\_rast WHERE rid=1) As foo;

convhull

|

env

--------------------------------------------------------+------------------------------------ ←

POLYGON((0.5 0.5,20.5 1.5,22.5 61.5,2.5 60.5,0.5 0.5)) | POLYGON((0 0,22 0,22 61,0 61,0 0) ←

)

**相关请参考**

ST\_Envelope, ST\_MinConvexHull, ST\_ConvexHull, ST\_AsText



PostGIS 2.2.0dev 开发手册

518 / 686

**9.11.4.3**

**ST\_DumpAsPolygons**

ST\_DumpAsPolygons —根据一个栅格波段，返回一个geomval (geom,val) 记录集。如果没有指定波段号，那么默认取波段1。

**用法**

setof geomval ST\_DumpAsPolygons(raster rast, integer band\_num=1, boolean exclude\_nodata\_value=TRUE);

**描述**

这是一个返回记录集的函数（SRF）。它返回geomval（类似键值对）记录集。每一个多边形都是有相同像素值的波段的像素并集。函数ST\_DumpAsPolygon对于将栅格多边形化是非常有用的，它是相当于GROUP BY(聚合)的逆过程。

例如，这个函数可以用于把单个栅格展开成多个几何对象结构，这些几何对象可以是：POLYGONS/MULTIPOLYGONS

可用版本：需要 GDAL版本 1.7或更高

**注意**

如果一个波段中有一个NODATA值的像素集，那么包含NODATA值的像素将不会被返回。

**注意**

如果你只关注一个栅格中给定值的像素的个数，使用函数ST\_ValueCount更快。

**注意**

这个函数和函数ST\_PixelAsPolygons不同，后者不论像素的像素值是什么都会返回一个几何对象。

**样例**

SELECT val, ST\_AsText(geom) As geomwkt

FROM (

SELECT (ST\_DumpAsPolygons(rast)).\*

FROM dummy\_rast

WHERE rid = 2

) As foo

WHERE val BETWEEN 249 and 251

ORDER BY val;

val |

geomwkt

-----+--------------------------------------------------------------------------

249 | POLYGON((3427927.95 5793243.95,3427927.95 5793243.85,3427928 5793243.85,

3427928 5793243.95,3427927.95 5793243.95))

250 | POLYGON((3427927.75 5793243.9,3427927.75 5793243.85,3427927.8 5793243.85,

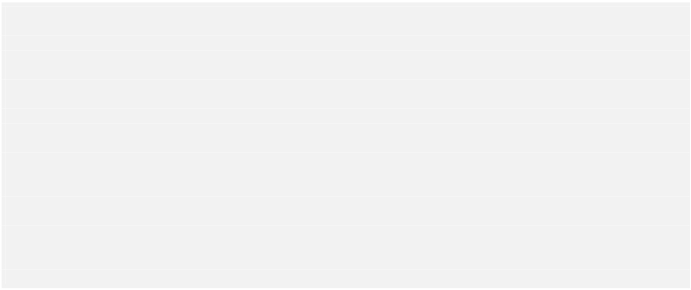
3427927.8 5793243.9,3427927.75 5793243.9))

250 | POLYGON((3427927.8 5793243.8,3427927.8 5793243.75,3427927.85 5793243.75,

3427927.85 5793243.8, 3427927.8 5793243.8))

251 | POLYGON((3427927.75 5793243.85,3427927.75 5793243.8,3427927.8 5793243.8,

3427927.8 5793243.85,3427927.75 5793243.85))



PostGIS 2.2.0dev 开发手册

519 / 686

**相关请参考**

geomval, ST\_Value, ST\_Polygon, ST\_ValueCount

**9.11.4.4**

**ST\_Envelope**

ST\_Envelope — 返回一个多边形用于表示栅格的边界。

**用法**

geometry ST\_Envelope(raster rast);

**描述**

返回一个多边形用于表示栅格的边界。它是一个float8精度的缓冲区，表示一个多边形。多边形是由缓冲区的各个顶点来确定的

((MINX, MINY), (MINX, MAXY), (MAXX, MAXY), (MAXX, MINY),(MINX, MINY))

**样例**

SELECT rid, ST\_AsText(ST\_Envelope(rast)) As envgeomwkt

FROM dummy\_rast;

rid |

envgeomwkt

-----+--------------------------------------------------------------------

1 | POLYGON((0 0,20 0,20 60,0 60,0 0))

2 | POLYGON((3427927 5793243,3427928 5793243,

3427928 5793244,3427927 5793244, 3427927 5793243))

**相关请参考**

ST\_Envelope, ST\_AsText, ST\_SRID

**9.11.4.5**

**ST\_MinConvexHull**

ST\_MinConvexHull —返回不包含NODATA像素的栅格的凸包几何对象

**用法**

geometry ST\_MinConvexHull(raster rast, integer nband=NULL);

**描述**

返回不包含NODATA像素的栅格的凸包几何对象。如果参数nband值为NULL，那么所有的栅格都会被处理。

可用版本：2.1.0

PostGIS 2.2.0dev 开发手册

520 / 686

**样例**

WITH foo AS (

SELECT

ST\_SetValues(

ST\_SetValues(

ST\_AddBand(ST\_AddBand(ST\_MakeEmptyRaster(9, 9, 0, 0, 1, -1, 0, 0, 0), 1, ’8BUI’, 0, ←

0), 2, ’8BUI’, 1, 0),

1, 1, 1,

ARRAY[

[0, 0, 0, 0, 0, 0, 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, 0, 0, 0, 0, 1],

[0, 0, 0, 1, 1, 0, 0, 0, 0],

[0, 0, 0, 1, 0, 0, 0, 0, 0],

[0, 0, 0, 0, 0, 0, 0, 0, 0],

[0, 0, 0, 0, 0, 0, 0, 0, 0],

[0, 0, 0, 0, 0, 0, 0, 0, 0]

]::double precision[][]

),

2, 1, 1,

ARRAY[

[0, 0, 0, 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, 0, 0, 0, 0, 1, 0, 0, 0],

[0, 0, 0, 0, 1, 1, 0, 0, 0],

[0, 0, 0, 0, 0, 1, 0, 0, 0],

[0, 0, 0, 0, 0, 0, 0, 0, 0],

[0, 0, 0, 0, 0, 0, 0, 0, 0],

[0, 0, 1, 0, 0, 0, 0, 0, 0]

]::double precision[][]

) AS rast

)

SELECT

ST\_AsText(ST\_ConvexHull(rast)) AS hull,

ST\_AsText(ST\_MinConvexHull(rast)) AS mhull,

ST\_AsText(ST\_MinConvexHull(rast, 1)) AS mhull\_1,

ST\_AsText(ST\_MinConvexHull(rast, 2)) AS mhull\_2

FROM foo

hull

mhull\_1

|

|

mhull

mhull\_2

|

←

----------------------------------+-------------------------------------+----------------------------

POLYGON((0 0,9 0,9 -9,0 -9,0 0)) | POLYGON((0 -3,9 -3,9 -9,0 -9,0 -3)) | POLYGON((3 -3,9

-3,9 -6,3 -6,3 -3)) | POLYGON((0 -3,6 -3,6 -9,0 -9,0 -3))

**相关请参考**

ST\_Envelope, ST\_ConvexHull, ST\_ConvexHull, ST\_AsText

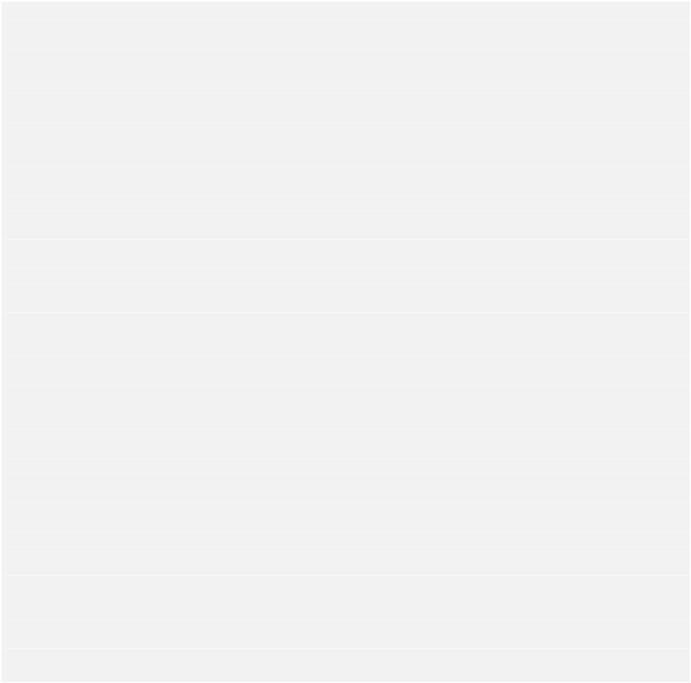
←

**9.11.4.6**

**ST\_Polygon**

ST\_Polygon — 返回一个multipolygon（多边形）几何对象，该几何对象是由那些像素值不是NODATA值的并集组成的。

如果没有指定是哪个波段，那么默认取波段1。



PostGIS 2.2.0dev 开发手册

521 / 686

**用法**

geometry ST\_Polygon(raster rast, integer band\_num=1);

**描述**

可用版本：0.1.6 需要 GDAL版本 1.7 or higher.

版本提升： 2.1.0提高了性能(完全基于C重新实现该函数) ，并且保证返回的多边形几何是有效的。

版本变更：2.1.0在之前的版本会有时候返回polygon几何对象，现在改成总是返回multipolygon几何对象。

**样例**

-- by default no data band value is 0 or not set, so polygon will return a square polygon

SELECT ST\_AsText(ST\_Polygon(rast)) As geomwkt

FROM dummy\_rast

WHERE rid = 2;

geomwkt

--------------------------------------------

MULTIPOLYGON(((3427927.75 5793244,3427928 5793244,3427928 5793243.75,3427927.75

5793243.75,3427927.75 5793244)))

-- now we change the no data value of first band

UPDATE dummy\_rast SET rast = ST\_SetBandNoDataValue(rast,1,254)

WHERE rid = 2;

SELECt rid, ST\_BandNoDataValue(rast)

from dummy\_rast where rid = 2;

-- ST\_Polygon excludes the pixel value 254 and returns a multipolygon

SELECT ST\_AsText(ST\_Polygon(rast)) As geomwkt

FROM dummy\_rast

WHERE rid = 2;

geomwkt

---------------------------------------------------------

←

MULTIPOLYGON(((3427927.9 5793243.95,3427927.85 5793243.95,3427927.85 5793244,3427927.9

←

5793244,3427927.9 5793243.95)),((3427928 5793243.85,3427928 5793243.8,3427927.95

5793243.8,3427927.95 5793243.85,3427927.9 5793243.85,3427927.9 5793243.9,3427927.9

←

←

5793243.95,3427927.95 5793243.95,3427928 5793243.95,3427928 5793243.85)),((3427927.8

5793243.75,3427927.75 5793243.75,3427927.75 5793243.8,3427927.75 5793243.85,3427927.75

←

←

5793243.9,3427927.75 5793244,3427927.8 5793244,3427927.8 5793243.9,3427927.8

←

5793243.85,3427927.85 5793243.85,3427927.85 5793243.8,3427927.85 5793243.75,3427927.8

5793243.75)))

-- Or if you want the no data value different for just one time

SELECT ST\_AsText(

ST\_Polygon(

ST\_SetBandNoDataValue(rast,1,252)

)

) As geomwkt

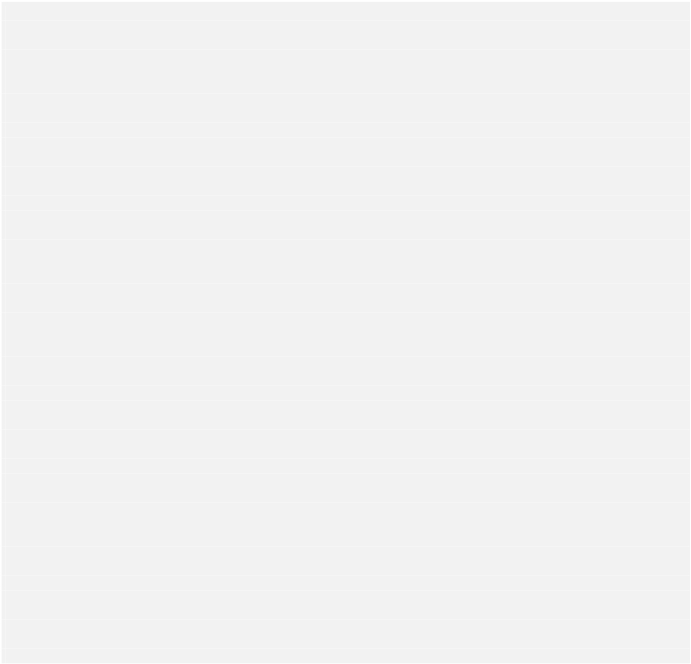
FROM dummy\_rast

WHERE rid =2;

geomwkt

---------------------------------

←



PostGIS 2.2.0dev 开发手册

522 / 686

MULTIPOLYGON(((3427928 5793243.85,3427928 5793243.8,3427928 5793243.75,3427927.85

←

5793243.75,3427927.8 5793243.75,3427927.8 5793243.8,3427927.75 5793243.8,3427927.75

←

5793243.85,3427927.75 5793243.9,3427927.75 5793244,3427927.8 5793244,3427927.85

←

5793244,3427927.9 5793244,3427928 5793244,3427928 5793243.95,3427928 5793243.85) ←

,(3427927.9 5793243.9,3427927.9 5793243.85,3427927.95 5793243.85,3427927.95

5793243.9,3427927.9 5793243.9)))

**相关请参考**

ST\_Value, ST\_DumpAsPolygons

←

**9.12**

**9.12.1**

**栅格运算符**

**&&**

&& —如果输入对象A的 bounding box与输入对象B的bounding box有交集，则返回TRUE

**用法**

boolean &&( raster A , raster B );

**描述**

如果输入栅格A的 bounding box与输入栅格B的bounding box有交集，则返回TRUE

**注意**

这个函数会充分利用栅格上的任意索引。

**样例**

SELECT A.rid As a\_rid, B.rid As b\_rid, A.rast && B.rast As intersect

FROM dummy\_rast AS A CROSS JOIN dummy\_rast AS B LIMIT 3;

a\_rid | b\_rid | intersect

-------+-------+---------

2 | 2 | t

2 | 3 | f

2 | 1 | f

**9.12.2**

**&<**

&< —如果栅格A的缓冲区在B的左边，那么返回true

**用法**

boolean &<( raster A , raster B );



PostGIS 2.2.0dev 开发手册

523 / 686

**描述**

如果输入栅格A的bounding box与输入栅格B的bounding box重合或者在后者的右边，则返回TRUE。或者更准确地说，

覆盖或者不在栅格B的bounding box的右侧。

**注意**

这个函数会充分利用几何对象上的任意索引。

**样例**

SELECT A.rid As a\_rid, B.rid As b\_rid, A.rast &< B.rast As overleft

FROM dummy\_rast AS A CROSS JOIN dummy\_rast AS B;

a\_rid | b\_rid | overleft

------+-------+----------

2 | 2 | t

2 | 3 | f

2 | 1 | f

3 | 2 | t

3 | 3 | t

3 | 1 | f

1 | 2 | t

1 | 3 | t

1 | 1 | t

**9.12.3**

**&>**

&> —如果A的bounding box覆盖或在B的bounding box的右侧，则返回TRUE

**用法**

boolean &>( raster A , raster B );

**描述**

如果A的bounding box覆盖或在B的bounding box的右侧，则返回TRUE，或者更准确地说覆盖或者不在栅格B的

bounding box的左侧

**注意**

这个函数会充分利用几何对象上的任意索引。

**样例**

SELECT A.rid As a\_rid, B.rid As b\_rid, A.rast &> B.rast As overright

FROM dummy\_rast AS A CROSS JOIN dummy\_rast AS B;

a\_rid | b\_rid | overright

-------+-------+----------



PostGIS 2.2.0dev 开发手册

524 / 686

2

2

2

3

3

3

1

1

1

|

|

|

|

|

|

|

|

|

2

3

1

2

3

1

2

3

1

|

|

|

|

|

|

|

|

|

t

t

t

f

t

f

f

t

t

**9.12.4**

**=**

= ——如果栅格A的bounding box与B的bounding box相同，则返回TRUE。注意：输入的栅格A和B的bounding box都是双精度

**用法**

boolean =( raster A , raster B );

**描述**

如果geometry或geography对象A的bounding box与geometry或geography对象B的bounding box相同，则返回TRUE。PostgreSQL

使用针对栅格定义的运算符=，<和>来进行内部排序和栅格之间的比较（例如在GROUP BY或ORDER BY这样的SQL子

句中）。

**警告**

这个运算符不会使用栅格上的任何索引，使用运算符~=代替，这个运算符可以用于栅格列的聚合操作。

可用版本：2.1.0

**相关请参考**

~=

**9.12.5**

**~=**

~= —如果栅格A的缓冲区和栅格B的缓冲区相同，则返回true。

**用法**

boolean ~=( raster A , raster B );

**描述**

如果栅格A的缓冲区和栅格B的缓冲区相同，则返回true。

**注意**

这个函数会充分利用栅格上的任意索引。

可用版本：2.0.0



PostGIS 2.2.0dev 开发手册

525 / 686

**样例**

非常有用的用例是：以两套单波段栅格，但每一个栅格代表不同的主题，用来创建一个多波段的栅格。

SELECT ST\_AddBand(prec.rast, alt.rast) As new\_rast

FROM prec INNER JOIN alt ON (prec.rast ~= alt.rast);

**相关请参考**

ST\_AddBand, =

**9.13**

**9.13.1**

**栅格和栅格波段空间关系函数**

**ST\_Contains**

ST\_Contains —当且仅当栅格rastB的所有点没有在栅格rastA的外部，并且rastB最少有一个点在rastA的内部（译者注：意思A把B完全包含了）。

**用法**

boolean ST\_Contains( raster rastA , integer nbandA , raster rastB , integer nbandB );

boolean ST\_Contains( raster rastA , raster rastB );

**描述**

当且仅当栅格rastB的所有点没有在栅格rastA的外部，并且rastB最少有一个点在rastA的内部（译者注：意思A把B完全包含了）。如果没有指定是哪个波段（或设置为NULL），只有栅格的凸包会被考虑，如果提供了波段号，只有值不为NODATA的像素会被考虑处理。

**注意**

这个函数会充分利用栅格上的任意索引。

**注意**

要想测试一个栅格和一个几何对象的空间关系，对栅格使用函数ST\_Polygon，

ST\_Contains(ST\_Polygon(raster), geometry) 或 ST\_Contains(geometry, ST\_Polygon(raster)).

**注意**

函数ST\_Contains()是函数ST\_Within()相反效果的函数，因此函数ST\_Contains(rastA, rastB) 的结果和函数ST\_Within(rastB, rastA)的结果一样。

可用版本：2.1.0



PostGIS 2.2.0dev 开发手册

526 / 686

**样例**

-- specified band numbers

SELECT r1.rid, r2.rid, ST\_Contains(r1.rast, 1, r2.rast, 1) FROM dummy\_rast r1 CROSS JOIN

dummy\_rast r2 WHERE r1.rid = 1;

←

NOTICE:

The first raster provided has no bands

rid | rid | st\_contains

-----+-----+-------------

1 | 1 |

1 | 2 | f

-- no band numbers specified

SELECT r1.rid, r2.rid, ST\_Contains(r1.rast, r2.rast) FROM dummy\_rast r1 CROSS JOIN

dummy\_rast r2 WHERE r1.rid = 1;

rid | rid | st\_contains

-----+-----+-------------

1 | 1 | t

1 | 2 | f

**相关请参考**

ST\_Intersects, ST\_Within

←

**9.13.2**

**ST\_ContainsProperly**

ST\_ContainsProperly —如果输入栅格B和栅格A的内部相交，但不和A的边界（或外部）有接触，那么返回TRUE

**用法**

boolean ST\_ContainsProperly( raster rastA , integer nbandA , raster rastB , integer nbandB );

boolean ST\_ContainsProperly( raster rastA , raster rastB );

**描述**

如果输入栅格B和栅格A的内部相交，但不和A的边界（或外部）有接触，那么返回TRUE。如果没有指定波段（或者设定为NULL值），那么该函数只适用于凸包型的栅格，如果指定了波段，那么只有有像元值（非NODATA）的栅格适用于这个函数。

**注意**

这个函数会充分利用栅格上的任意索引。

**注意**

要想测试一个栅格和一个几何对象之间的关系，对栅格应用函数ST\_Polygon，例如ST\_ContainsProperly(ST\_Polygon(raster), geometry) 或ST\_ContainsProperly(geometry, ST\_Polygon(raster)).

可用版本：2.1.0



PostGIS 2.2.0dev 开发手册

527 / 686

**样例**

SELECT r1.rid, r2.rid, ST\_ContainsProperly(r1.rast, 1, r2.rast, 1) FROM dummy\_rast r1 CROSS ←

JOIN dummy\_rast r2 WHERE r1.rid = 2;

rid | rid | st\_containsproperly

-----+-----+---------------------

2 | 1 | f

2 | 2 | f

**相关请参考**

ST\_Intersects, ST\_Contains

**9.13.3**

**ST\_Covers**

ST\_Covers —如果栅格rastB没有点在栅格rastA外，则返回true。

**用法**

boolean ST\_Covers( raster rastA , integer nbandA , raster rastB , integer nbandB );

boolean ST\_Covers( raster rastA , raster rastB );

**描述**

如果栅格rastB没有点在栅格rastA外，则返回true。如果没有指定波段（或者设定为NULL值），那么该函数只适用于凸包型的栅格，如果指定了波段，那么只有有像元值（非NODATA）的栅格适用于这个函数。

**注意**

这个函数会充分利用栅格上的任意索引。

**注意**

想要测试一个栅格和一个几何对象的空间关系，对栅格使用函数ST\_Polygon，例如ST\_Covers(ST\_Polygon(raster), geometry) 或 ST\_Covers(geometry, ST\_Polygon(raster)).

可用版本：2.1.0

**样例**

SELECT r1.rid, r2.rid, ST\_Covers(r1.rast, 1, r2.rast, 1) FROM dummy\_rast r1 CROSS JOIN

dummy\_rast r2 WHERE r1.rid = 2;

rid | rid | st\_covers

-----+-----+-----------

2 | 1 | f

2 | 2 | t

←



PostGIS 2.2.0dev 开发手册

528 / 686

**相关请参考**

ST\_Intersects, ST\_CoveredBy

**9.13.4**

**ST\_CoveredBy**

ST\_CoveredBy —如果栅格rastA没有点在栅格rastB外，则返回true。

**用法**

boolean ST\_CoveredBy( raster rastA , integer nbandA , raster rastB , integer nbandB );

boolean ST\_CoveredBy( raster rastA , raster rastB );

**描述**

如果栅格rastA没有点在栅格rastB外，则返回true。如果没有指定波段（或者设定为NULL值），那么该函数只适用于凸包型的栅格，如果指定了波段，那么只有有像元值（非NODATA）的栅格适用于这个函数。

**注意**

这个函数会充分利用栅格上的任意索引。

**注意**

想要测试一个栅格和一个几何对象的空间关系，对栅格使用函数ST\_Polygon，例如ST\_CoveredBy(ST\_Polygon(raster), geometry) 或 ST\_CoveredBy(geometry, ST\_Polygon(raster)).

可用版本：2.1.0

**样例**

SELECT r1.rid, r2.rid, ST\_CoveredBy(r1.rast, 1, r2.rast, 1) FROM dummy\_rast r1 CROSS JOIN

dummy\_rast r2 WHERE r1.rid = 2;

rid | rid | st\_coveredby

-----+-----+--------------

2 | 1 | f

2 | 2 | t

**相关请参考**

ST\_Intersects, ST\_Covers

←

**9.13.5**

**ST\_Disjoint**

ST\_Disjoint —如果两个栅格rastA和栅格rastB没有空间相交，则返回true。



PostGIS 2.2.0dev 开发手册

529 / 686

**用法**

boolean ST\_Disjoint( raster rastA , integer nbandA , raster rastB , integer nbandB );

boolean ST\_Disjoint( raster rastA , raster rastB );

**描述**

如果两个栅格rastA和栅格rastB没有空间相交，则返回true。如果没有指定波段（或者设定为NULL值），那么该函数只适用于凸包型的栅格，如果指定了波段，那么只有有像元值（非NODATA）的栅格适用于这个函数。

**注意**

该函数不使用任何索引。

**注意**

想要测试一个栅格和一个几何对象的空间关系，对栅格使用函数ST\_Polygon，例如

ST\_Disjoint(ST\_Polygon(raster), geometry).

可用版本：2.1.0

**样例**

-- rid = 1 has no bands, hence the NOTICE and the NULL value for st\_disjoint

SELECT r1.rid, r2.rid, ST\_Disjoint(r1.rast, 1, r2.rast, 1) FROM dummy\_rast r1 CROSS JOIN

dummy\_rast r2 WHERE r1.rid = 2;

←

NOTICE:

The second raster provided has no bands

rid | rid | st\_disjoint

-----+-----+-------------

2 | 1 |

2 | 2 | f

-- this time, without specifying band numbers

SELECT r1.rid, r2.rid, ST\_Disjoint(r1.rast, r2.rast) FROM dummy\_rast r1 CROSS JOIN

dummy\_rast r2 WHERE r1.rid = 2;

rid | rid | st\_disjoint

-----+-----+-------------

2 | 1 | t

2 | 2 | f

**相关请参考**

ST\_Intersects

←

**9.13.6**

**ST\_Intersects**

ST\_Intersects — 如果栅格rastA和栅格rastB空间上相交，则返回true



PostGIS 2.2.0dev 开发手册

530 / 686

**用法**

boolean ST\_Intersects( raster rastA , integer nbandA , raster rastB , integer nbandB );

boolean ST\_Intersects( raster rastA , raster rastB );

boolean ST\_Intersects( raster rast , integer nband , geometry geommin );

boolean ST\_Intersects( raster rast , geometry geommin , integer nband=NULL );

boolean ST\_Intersects( geometry geommin , raster rast , integer nband=NULL );

**描述**

如果栅格rastA和栅格rastB空间上相交，则返回true。如果没有指定波段（或者设定为NULL值），那么该函数只适用于凸包型的栅格，如果指定了波段，那么只有有像元值（非NODATA）的栅格适用于这个函数。

**注意**

这个函数会充分利用栅格上的任意索引。

版本提升： 2.0.0 引入支持栅格之间的相交

**警告**

版本变更： 2.1.0 函数变体ST\_Intersects(raster, geometry) 的结果返回已经改变了以便匹配ST\_Intersects(geometry, raster)的返回结果

**样例**

-- different bands of same raster

SELECT ST\_Intersects(rast, 2, rast, 3) FROM dummy\_rast WHERE rid = 2;

st\_intersects

---------------

t

**相关请参考**

ST\_Intersection, ST\_Disjoint

**9.13.7**

**ST\_Overlaps**

ST\_Overlaps —如果栅格rastA和rastB相交，但是一个并没有完全包含另一个，则返回true。

**用法**

boolean ST\_Overlaps( raster rastA , integer nbandA , raster rastB , integer nbandB );

boolean ST\_Overlaps( raster rastA , raster rastB );



PostGIS 2.2.0dev 开发手册

531 / 686

**描述**

如果栅格rastA和rastB相交，但是一个并没有完全包含另一个，则返回true。

如果没有指定波段（或者设定为NULL值），那么该函数只适用于凸包型的栅格，如果指定了波段，那么只有有像元值（非NODATA）的栅格适用于这个函数。

**注意**

这个函数会充分利用栅格上的任意索引。

**注意**

想要测试一个栅格和一个几何对象的空间关系，在栅格上使用函数ST\_Polygon

ST\_Overlaps(ST\_Polygon(raster), geometry).

可用版本：2.1.0

**样例**

-- comparing different bands of same raster

SELECT ST\_Overlaps(rast, 1, rast, 2) FROM dummy\_rast WHERE rid = 2;

st\_overlaps

-------------

f

**相关请参考**

ST\_Intersects

**9.13.8**

**ST\_Touches**

ST\_Touches — 如果两个栅格rastA和rastB有至少一个共同点，但内部并不相交，那么返回true。

**用法**

boolean ST\_Touches( raster rastA , integer nbandA , raster rastB , integer nbandB );

boolean ST\_Touches( raster rastA , raster rastB );

**描述**

如果两个栅格rastA和rastB有至少一个共同点，但内部并不相交，那么返回true。如果没有指定波段（或者设定为NULL值），那么该函数只适用于凸包型的栅格，如果指定了波段，那么只有有像元值（非NODATA）的栅格适用于这个函数。

**注意**

这个函数会充分利用栅格上的任意索引。



PostGIS 2.2.0dev 开发手册

532 / 686

**注意**

想要测试一个栅格和一个几何对象的空间关系，在栅格上使用函数ST\_Polygon，例如ST\_Touches(ST\_Polygon(raster), geometry).

可用版本：2.1.0

**样例**

SELECT r1.rid, r2.rid, ST\_Touches(r1.rast, 1, r2.rast, 1) FROM dummy\_rast r1 CROSS JOIN

dummy\_rast r2 WHERE r1.rid = 2;

rid | rid | st\_touches

-----+-----+------------

2 | 1 | f

2 | 2 | f

**相关请参考**

ST\_Intersects

←

**9.13.9**

**ST\_SameAlignment**

ST\_SameAlignment — 如果上之间有相同的特征值skew, scale, spatial ref （即SRID），则返回true。如果有不同，则返回false。

**用法**

boolean ST\_SameAlignment( raster rastA , raster rastB );

boolean ST\_SameAlignment( double precision ulx1 , double precision uly1 , double precision scalex1 , double precision scaley1

, double precision skewx1 , double precision skewy1 , double precision ulx2 , double precision uly2 , double precision scalex2 ,

double precision scaley2 , double precision skewx2 , double precision skewy2 );

boolean ST\_SameAlignment( raster set rastﬁeld );

**描述**

非聚合函数版本 (函数变体1和函数变体2): 如果两个栅格（或者直接提供或者使用参数upperleft, scale, skew 和 srid的值）有相同的参数scale, skew, srid 和至少有一个栅格的任意像素的任意角点和另一个栅格的任意像素的任意角点重叠，那么返回true。如果不满足这些条件将会返回false，并提示对齐方式的细节问题。

聚合版本 (函数变体3)：如果一个集合的栅格都aligned（对齐），那么返回true。函数ST\_SameAlignment()从PostgreSQL的术语来看是一个“聚合”函数。这意味着它的执行方式和SQL里面的函数SUM() 和 AVG() 是类似的。

可用版本：2.0.0

版本提升： 2.1.0 添加聚合版本的函数变体

**样例: Rasters**



PostGIS 2.2.0dev 开发手册

533 / 686

SELECT ST\_SameAlignment(

ST\_MakeEmptyRaster(1, 1, 0, 0, 1, 1, 0, 0),

ST\_MakeEmptyRaster(1, 1, 0, 0, 1, 1, 0, 0)

) as sm;

sm

----

t

SELECT ST\_SameAlignment(A.rast,b.rast)

FROM dummy\_rast AS A CROSS JOIN dummy\_rast AS B;

NOTICE:

NOTICE:

The two rasters provided have different SRIDs

The two rasters provided have different SRIDs

st\_samealignment

------------------

t

f

f

f

**相关请参考**

Section 5.1, ST\_NotSameAlignmentReason, ST\_MakeEmptyRaster

**9.13.10**

**ST\_NotSameAlignmentReason**

ST\_NotSameAlignmentReason — 如果栅格对齐了那么返回描述信息，如果没有对齐，返回没有对齐的原因。

**用法**

boolean ST\_SameAlignment(raster rastA, raster rastB);

**描述**

如果栅格对齐了那么返回描述信息，如果没有对齐，返回没有对齐的原因。

**注意**

如果栅格没有对齐有多种原因，那么只返回其中一个原因（第一个让对齐失败的理由）。

可用版本：2.1.0

**样例**

SELECT

ST\_SameAlignment(

ST\_MakeEmptyRaster(1, 1, 0, 0, 1, 1, 0, 0),

ST\_MakeEmptyRaster(1, 1, 0, 0, 1.1, 1.1, 0, 0)

),

ST\_NotSameAlignmentReason(

ST\_MakeEmptyRaster(1, 1, 0, 0, 1, 1, 0, 0),



PostGIS 2.2.0dev 开发手册

534 / 686

ST\_MakeEmptyRaster(1, 1, 0, 0, 1.1, 1.1, 0, 0)

)

;

st\_samealignment |

st\_notsamealignmentreason

------------------+-------------------------------------------------

f

| The rasters have different scales on the X axis

(1 row)

**相关请参考**

Section 5.1, ST\_SameAlignment

**9.13.11**

**ST\_Within**

ST\_Within — 当且仅当栅格rastA的所有点没有在栅格rastB的外部，并且rastA最少有一个点在rastB的内部（译者注：意思A把B完全包含了）。

**用法**

boolean ST\_Within( raster rastA , integer nbandA , raster rastB , integer nbandB );

boolean ST\_Within( raster rastA , raster rastB );

**描述**

当且仅当栅格rastA的所有点没有在栅格rastB的外部，并且rastA最少有一个点在rastB的内部（译者注：意思A把B完全包含了）。如果没有指定波段（或者设定为NULL值），那么该函数只适用于凸包型的栅格，如果指定了波段，那么只有有像元值（非NODATA）的栅格适用于这个函数。

**注意**

这个函数会充分利用栅格上的任意索引。

**注意**

要想测试一个栅格和一个几何对象的空间关系，对栅格使用函数ST\_Polygon，例如ST\_Within(ST\_Polygon(raster), geometry) 或 ST\_Within(geometry, ST\_Polygon(raster)).

**注意**

函数ST\_Within() 是函数ST\_Contains()的逆实现，因此函数ST\_Within(rastA, rastB)与函数ST\_Contains(rastB, rastA)调用结果相同。

可用版本：2.1.0



PostGIS 2.2.0dev 开发手册

535 / 686

**样例**

SELECT r1.rid, r2.rid, ST\_Within(r1.rast, 1, r2.rast, 1) FROM dummy\_rast r1 CROSS JOIN

dummy\_rast r2 WHERE r1.rid = 2;

rid | rid | st\_within

-----+-----+-----------

2 | 1 | f

2 | 2 | t

**相关请参考**

ST\_Intersects, ST\_Contains, ST\_DWithin, ST\_DFullyWithin

←

**9.13.12**

**ST\_DWithin**

ST\_DWithin — 如果栅格rastA和栅格rastB的距离在指定的距离内，则返回true。

**用法**

boolean ST\_DWithin( raster rastA , integer nbandA , raster rastB , integer nbandB , double precision distance\_of\_srid );

boolean ST\_DWithin( raster rastA , raster rastB , double precision distance\_of\_srid );

**描述**

如果栅格rastA和栅格rastB的距离在指定的距离内，则返回true。如果没有指定波段（或者设定为NULL值），那么该函数只适用于凸包型的栅格，如果指定了波段，那么只有有像元值（非NODATA）的栅格适用于这个函数。距离单位是由栅格所在的空间参考系所定义的。要想使用这个函数，输入的源栅格必须都有相同的坐标参考系也就是有相同的SRID。

**注意**

这个函数会充分利用栅格上的任意索引。

**注意**

想要测试栅格和几何对象之间的关系，需要对栅格使用函数ST\_Polygon，例如

ST\_DWithin(ST\_Polygon(raster), geometry).

可用版本：2.1.0

**样例**

SELECT r1.rid, r2.rid, ST\_DWithin(r1.rast, 1, r2.rast, 1, 3.14) FROM dummy\_rast r1 CROSS

JOIN dummy\_rast r2 WHERE r1.rid = 2;

rid | rid | st\_dwithin

-----+-----+------------

2 | 1 | f

2 | 2 | t

←



PostGIS 2.2.0dev 开发手册

536 / 686

**相关请参考**

ST\_Within, ST\_DFullyWithin

**9.13.13**

**ST\_DFullyWithin**

ST\_DFullyWithin —如果栅格rastA和栅格rastB的距离完全在指定的距离内，则返回true。

**用法**

boolean ST\_DFullyWithin( raster rastA , integer nbandA , raster rastB , integer nbandB , double precision distance\_of\_srid );

boolean ST\_DFullyWithin( raster rastA , raster rastB , double precision distance\_of\_srid );

**描述**

如果栅格rastA和栅格rastB的距离完全在指定的距离内，则返回true。如果没有指定波段（或者设定为NULL值），那么该函数只适用于凸包型的栅格，如果指定了波段，那么只有有像元值（非NODATA）的栅格适用于这个函数。距离单位是由栅格所在的空间参考系所定义的。要想使用这个函数，输入的源栅格必须都有相同的坐标参考系也就是有相同的SRID。

**注意**

这个函数会充分利用栅格上的任意索引。

**注意**

想要测试栅格和几何对象之间的关系，需要在栅格上使用函数ST\_Polygon，例如：

ST\_DFullyWithin(ST\_Polygon(raster), geometry).

可用版本：2.1.0

**样例**

SELECT r1.rid, r2.rid, ST\_DFullyWithin(r1.rast, 1, r2.rast, 1, 3.14) FROM dummy\_rast r1

CROSS JOIN dummy\_rast r2 WHERE r1.rid = 2;

rid | rid | st\_dfullywithin

-----+-----+-----------------

2 | 1 | f

2 | 2 | t

**相关请参考**

ST\_Within, ST\_DWithin

←



PostGIS 2.2.0dev 开发手册

537 / 686

**Chapter 10**

**PostGIS 栅格常见问题**

1. 我在哪里可以找到更多关于PostGIS栅格项目的信息呢？

参考 [PostGIS Raster home page.](http://trac.osgeo.org/postgis/wiki/WKTRaster)

2. 有没有一些书籍或者入门教程能够让我入门这个很棒的发明？

答：这里有一个完整的初学者指南： Intersecting vector buffers with large raster coverage using PostGIS Raster(http://trac.osgeo.org/postgis/wiki/WKTRasterTutorial01). Jorge有一系列关于PostGIS栅格的博客，包含了展示怎样将栅格数据加载到Oracle GeoRaster. 请查看Jorge’s PostGIS Raster / Oracle GeoRaster Series.（http://gis4free.wordpress.com/category/postgis-raster/）。在 [PostGIS in Action - Rasterchapter](http://www.postgis.us/chapter_13)（http://www.postgis.us/page\_buy\_book）有一个完整的（超过35页内容）介绍PostGIS栅格，还有免费的代码和数据下载。你可以在Manning购买PostGIS in Action这本书（批量购买有大折扣哦）或者值购买这本书的电子格式。你同样可以从Amazon或者其他渠道购买。所有的纸质书籍都提供免费的电子书版本。下面是一个PostGIS 栅格用户的评论：

[PostGIS raster applied to landclassiﬁcation urban forestry](http://fuzzytolerance.info/code/postgis-raster-ftw/)（译者注：这个链接已经打不开了，原文如此，真够操蛋）

3. 我该怎样在我的PostGIS数据库中安装栅格支持？

答：最简单的办法是下周PostGIS的二进制包，当前已经可以使用Windows和最新版本的Mac OSX系统中使用栅格功能。首先你需要一个可以正常运行的PostGIS 2.0.0或以上版本以及正常运行的PostgreSQL 8.4, 9.0, 或 9.1版本。在PostGIS2.0中已经集成了PostGIS Raster栅格功能，因此在你编译安装PostGIS时候，也就同时编译了栅格模块。Windows下安装和运行PostGIS请参考How to Install and Conﬁgure PostGIS raster on windows(http://gis4free.wordpress.com/2011/03/10/how-to-install-and-configure-postgis-raster-on-windows/)如果你在Windows上运行，你可以自己编译或者使用预编译包pre-compiled PostGIS Raster windows binaries（http://gis4free.wordpress.com/2011/03/10/how-to-install-and-configure-postgis-raster-on-windows/）. 如果你使用的是Mac OSX系统Leopard 或 SnowLeopard，二进制包可以在这里获取Kyng Chaos Mac OSX PostgreSQL/GIS binaries(http://www.kyngchaos.com/software/postgres)。然后在你的数据库中开启栅格功能支持，也就是在数据库中运行脚本rtpostgis.sql 。若想进行PostGIS升级，使用脚本rtpostgis\_upgrade\_minor..sql而不是脚本rtpostgis.sql。对于其他平台你可以自行编译，所需要的依赖是PostGIS 和GDAL库。更多关于使用源码进行编译，请参考 [Installing PostGIS Raster from source (in prior versions ofPostGIS)](http://trac.osgeo.org/postgis/wiki/WKTRaster/Documentation01#a2.3-CompilingandInstallingfromSources)（http://trac.osgeo.org/postgis/wiki/WKTRaster/Documentation01#a2.3-CompilingandInstallingfromSources）。

4. 我在Linux上面尝试运行脚本rtpostgis.sql时候碰到一个加载库的错误："C:/Program Files/PostgreSQL/8.4/lib/rtpostgis.dll": The speciﬁed module could not befound. or could not load library

答：rtpostgis.so/dll库的构建依赖于库libgdal.dll/so。确认你在Windows上面在你的PostgreSQL安装目录的bin目录下有库libgdal-1.dll。对于Linux平台，也必须安装成功。如果你的数据库中没有安装PostGIS，运行该脚本也会得到不同的错误。请在尝试安装栅格功能支持前，确保在你的数据库中首先安装PostGIS完成。

5. 我怎样把栅格数据加载到PostGIS数据库中?

答：最新版本的PostGIS打包了栅格加载工具raster2pgsql，可以加载各种类型的栅格，不用附件软件也能生成栅格概览。请参考章节5.1.1获得更多详细信息。2.0版本之前的加载工具是raster2pgsql.py，需要Python的numpy模块和GDAL库，现在已经不再支持了。

6. 什么样的栅格文件格式我可以导入到数据库中？

答：任何GDAL库支持的格式都可以导入。GDAL支持格式列表如下： GDAL File Formats。适合你的GIS库的GDAL版本可能不一定支持所有的格式。想要验证你的GDAL版本支持的格式，你需要执行如下命令：

PostGIS 2.2.0dev 开发手册

538 / 686

raster2pgsql -G

7. 我能把PostGIS栅格数据导出能其他栅格格式吗？

答：当然可以。GDAL 1.7+有一个PostGIS栅格驱动，但是只有在编译时候选择了带有PostgreSQL支持才行。这个驱动版本当前不支持非常规大小的的栅格，虽然你可以把不规则块的栅格存储在PostGIS栅格类型。如果你使用源码编译，你需要加上如下编译选项：

--with-pg=path/to/pg\_config

来开启驱动的支持。参考GDAL Build Hints获取在不同操作系统平台上GDAL库依赖的建议。如果你的GDAL库添加了PostGIS 栅格驱动的支持，你可以在执行下面命令时候，查看到PostGIS 栅格格式列表：

gdalinfo --formats

想通过GDAL库获取关于你栅格的概要，使用gdalinfo命令：

gdalinfo

"PG:host=localhost port=5432 dbname=’mygisdb’ user=’postgres’ password=’ ←

whatever’ schema=’someschema’ table=sometable"

想要把栅格数据导出成其他栅格格式，使用下面的命令 gdal\_translate会把一个表的所有数据导出成PNG文件，文件大小是原有数据的10%。和所依赖的像素类型，如果导出格式不支持当前的像素类型，那么类型转换可能会不成功。例如对于float point band类型和32位unsigned int类型想要转换成JPG格式或其他格式就不那么容易。下面是一个简单样例：

gdal\_translate -of PNG -outsize 10% 10% "PG:host=localhost port=5432 dbname=’mygisdb’

←

user=’postgres’ password=’whatever’ schema=’someschema’ table=sometable" C:\ ←

somefile.png

你同样可以在导出时候，在内的驱动连接时候，使用SQL的where子句 where=…。下面使用了一个where子句：

gdal\_translate -of PNG -outsize 10% 10% "PG:host=localhost port=5432 dbname=’mygisdb’

user=’postgres’ password=’whatever’ schema=’someschema’ table=sometable where=’ ←

filename=\’abcd.sid\’’" " C:\somefile.png

gdal\_translate -of PNG -outsize 10% 10% "PG:host=localhost port=5432 dbname=’mygisdb’

←

←

user=’postgres’ password=’whatever’ schema=’someschema’ table=sometable where=’ ←

ST\_Intersects(rast, ST\_SetSRID(ST\_Point(-71.032,42.3793),4326) )’ " C:\ ←

intersectregion.png

更多样例和语法请参考： [Reading Raster Data of PostGIS Raster section](http://trac.osgeo.org/gdal/wiki/frmts_wtkraster.html#a3.2-Readingrasterdatafromthedatabase)

8. 有没有已经添加了PostGIS Raster栅格功能支持的GDAL库的二进制格式程序？

答：当然有的。请查看页面GDAL Binaries。任何添加了PostgreSQL支持的GDAL库都有PostGIS Raster功能支持。PostGIS Raster 经历了许多改变。如果你想获取到Windows版本最新的nightly build（每日构建版本）——那么你可以check 出Tamas Szekeres的使用Visual Studio 打包的包含了GDAL分支， Python 绑定信息和MapServer 可执行程序以及打包的PostGIS Raster栅格驱动。你只需要单机SDK脚本，然后执行命令即可。[http://vbkto.dyndns.org/sdk/.](http://vbkto.dyndns.org/sdk/)同样VS项目文件可以在这里获取到：[FWTools latest stable version for Windows is compiledwith Raster support.](http://fwtools.maptools.org/)

9. 我能用什么工具来查看PostGIS 的栅格数据？

答：你可以使用添加了GDAL 1.7+版本库和PostGIS Raster栅格驱动支持的MapServer来查看栅格数据。QuantumGIS

(QGIS) 现在支持查看PostGIS Raster栅格数据，如果已经安装了PostGIS 栅格驱动了的话；理论上任何使用GDAL库的工具都可以支持PostGIS 栅格数据或者以相当小的代价做支持。同样对于Windows平台，如果你不想自己找麻烦来编译自己的版本，那么Tamas的二进制包http://vbkto.dyndns.org/sdk/是一个非常好的选择。

PostGIS 2.2.0dev 开发手册

539 / 686

10. 我怎样把PostGIS 栅格数据层添加到我的MapServer 地图中？

答：首先你需要添加PostGIS栅格编译支持的GDAL 1.7 或更高版本的库。GDAL 1.8或更高版本是推荐版本，因为在GDAL1.8版本的库中修复了许多问题，在PostGIS分支中也修复了许多问题，这个可能会让你更喜欢。请参考MapServer Raster processing options获取不同你可以在MapServer栅格图层进行处理的函数。让我们对PostGIS 栅格数据特别感兴趣的是因为每个瓦片可以有不同标准的数据库列，你可以在你的数据源上做一些分割。下面是一个展示怎样在你的MapServer中定义PostGIS栅格图层的案例。

**注意**

对于瓦片化的栅格，参数mode=2 是需要的，并且在PostGIS 2.0 和 GDAL 1.8 驱动中都添加了支持。这个参数在GDAL 1.7驱动中已经添加了支持。

-- displaying raster with standard raster options

LAYER

NAME coolwktraster

TYPE raster

STATUS ON

DATA "PG:host=localhost port=5432 dbname=’somedb’ user=’someuser’ password=’whatever’

schema=’someschema’ table=’cooltable’ mode=’2’"

PROCESSING "NODATA=0"

PROCESSING "SCALE=AUTO"

#... other standard raster processing functions here

#... classes are optional but useful for 1 band data

CLASS

NAME "boring"

EXPRESSION ([pixel] < 20)

COLOR 250 250 250

END

CLASS

NAME "mildly interesting"

EXPRESSION ([pixel] > 20 AND [pixel] < 1000)

COLOR 255 0 0

END

CLASS

NAME "very interesting"

EXPRESSION ([pixel] >= 1000)

COLOR 0 255 0

END

END

-- displaying raster with standard raster options and a where clause

LAYER

NAME soil\_survey2009

TYPE raster

STATUS ON

DATA "PG:host=localhost port=5432 dbname=’somedb’ user=’someuser’ password=’whatever’

schema=’someschema’ table=’cooltable’ where=’survey\_year=2009’ mode=’2’"

PROCESSING "NODATA=0"

#... other standard raster processing functions here

#... classes are optional but useful for 1 band data

END

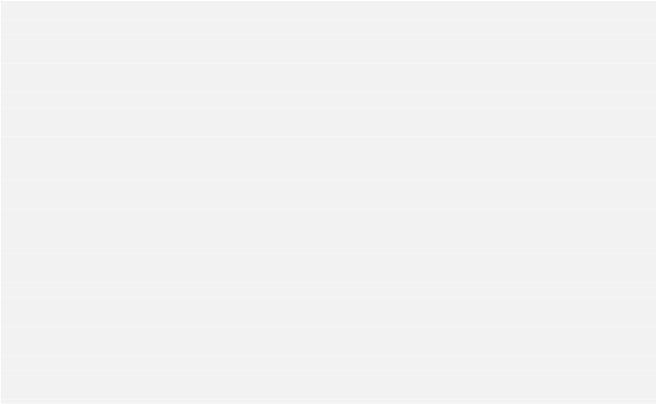
11. 我当前的栅格数据可以使用那些函数？

答：参考章节Chapter 9。还有更多的函数，但这些目前都在开发中。参考PostGIS的路线图 [PostGIS Raster roadmap page](http://trac.osgeo.org/postgis/wiki/WKTRaster/PlanningAndFunding)

获取将来你能得到的一些信息。

12.我碰到了一个错误ERROR: function st\_intersects(raster, unknown) is not unique or st\_union(geometry,text) is not unique.

我该怎么解决？



PostGIS 2.2.0dev 开发手册

540 / 686

这个错误出现的原因是你传递给函数的参数是一个几何对象的文本描述，而不是几何对象类型即geometry类型。在这种情况下，PostgreSQL会把这个文本表述当做一种未知类型，这意味着会它可以调用函数st\_intersects(raster, geometry) 或函数 st\_intersects(raster,raster) ，因此会导致不唯一的调用情况，因为理论上这两个函数都能支持你的调用请求。

要想避免这种情况，你需要把这种文本描述转换成一个几何对象。例如，你可以写成下面这种：

SELECT rast

FROM my\_raster

WHERE ST\_Intersects(rast, ’SRID=4326;POINT(-10 10)’);

Cast the textual geometry representation to a geometry by changing your code to this:

SELECT rast

FROM my\_raster

WHERE ST\_Intersects(rast, ’SRID=4326;POINT(-10 10)’::geometry);

13. PostGIS Raster的栅格特性从类型上看与Oracle GeoRaster (SDO\_GEORASTER) 和 SDO\_RASTER有什么不同？

答：关于这个话题的更多扩展性讨论，请查看Jorge Arévalo [Oracle GeoRaster and PostGIS Raster: Firstimpressions](http://gis4free.wordpress.com/2010/07/19/oracle-georaster-part-i/)。

one-georeference-by-raster模式对one-georeference-by-layer的主要优势是允许地域覆盖数据不必矩形化（这通常是栅格覆盖覆盖大型边界面临的问题，请参考文档中相关的栅格文档）。

第二：允许栅格交叠（这个对于实现矢量数据结构转换成栅格数据结构上是必要的），这些功能在Oracle也能实现，但是他们的实现方式是多个SDO\_GEORASTER对象的存储对应多个SDO\_RASTER表。一个复杂的地域覆盖会导致在数据库中创建上百个表。而使用PostGIS Raster栅格功能，你可以把相似的栅格存储在一个表中。

这有点像PostGIS强制你只存储完全的矩形化矢量的没有间隙或交叠的地域覆盖面。

(一个完美的矩形化的拓扑层). 这在一些应用中是非常实用的但是实践中却表明对于大多数的地理覆盖这种处理不现实也是不必要的。矢量结构需要弹性地存储不连续的和非矩形规则化的面。我们认为这是一个巨大的优势，栅格结构同样会因此而受益。

14. 程序raster2pgsql在加载大型文件时候报错并提示： String of N bytes is too long for encoding conversion？怎么回事？

答：程序raster2pgsql 在生成加载文件时候不做任何连接数据库的操作。如果你的数据库的客户端编码格式与数据库的编码格式不同，那么当加载大型栅格文件（大约30M）时候，你可能会遇到一个bytes is too long for encoding conversion 的错误。这种情况通常发生于，比如，你数据库使用UTF-8编码，但是为了支持Windows 应用，你把客户端编码设置为WIN1252。要想解决这个问题，就必须在加载数据时候把客户端编码和数据库使用编码设置相同。

例如，你可以在Windows上设置如下：

set PGCLIENTENCODING=UTF8

如果你使用的是Unix/Linux平台

export PGCLIENTENCODING=UTF8

该这个狗血问题的细节请参考<http://trac.osgeo.org/postgis/ticket/2209>

PostGIS 2.2.0dev 开发手册

541 / 686

**Chapter 11**

**拓扑**

PostGIS 的拓扑类型和函数用于管理拓扑结构，比如面、边界和点。

Sandro Santilli在2011年巴黎PostGIS大会上给出了PostGIS 拓扑功能非常好的演示。请前往如下地址查看[Topology with PostGIS 2.0 slide deck.](http://strk.keybit.net/projects/postgis/Paris2011_TopologyWithPostGIS_2_0.pdf)

Vincent Picavet 提供了关于拓扑简介和使用，以及不同的FOSS4G工具来支持拓扑功能

[State of the art of FOSS4G for topology and network analysis.](http://2010.foss4g.org/presentations/3555.pdf)

一个基于GIS数据库的拓扑样例请参考 [US Census Topologically Integrated Geographic Encoding and](http://www.census.gov/geo/www/tiger/overview.html)Ref[erence System (TIGER)](http://www.census.gov/geo/www/tiger/overview.html)。如果你想体验PostGIS拓扑功能和需要一些数据，请查看 Topology\_Load\_Tiger.

PostGIS拓扑模块在之前的版本中一直存在，但在PostGIS官方文档中却没有介绍。

在PostGIS 2.0.0中，主要的清理工作是移除一些不推荐的函数，修复已知的问题，更好的用文档方式介绍拓扑方面的特性和函数，增加新函数，以及增加对SQL-MM规范的支持

更多该项目的信息参考 [PostGIS Topology Wiki](http://trac.osgeo.org/postgis/wiki/UsersWikiPostgisTopology)

关于该模块的所有函数和表都安装在一个叫topology的schema中

遵守SQL/MM规范的函数都以ST\_为前缀，PostGIS自己的函数没有加上前缀

如果需要PostGIS 拓扑功能，需要像第二章介绍那样，在安装PostGIS时候添加编译选项 --with-topology 。一些函数依赖于

GEOS 3.3+ 版本，因此你在编译时候需要GEOS 3.3+版本支持，以便获取完整的拓扑支持

**11.1**

**11.1.1**

**拓扑类型**

**getfaceedges\_returntype**

getfaceedges\_returntype — 一种复杂的类型，包括一个序列号和一个边界号。这种类型是函数ST\_GetFaceEdges返回的。

**描述**

一种复杂的类型，包括一个序列号和一个边界号。这种类型是函数ST\_GetFaceEdges返回的。

1. sequence：是一个整型值。它引用定义在表topology.topology（该表定义拓扑的schema和SRID） 的一个拓扑结构。

2. edge：一个整型值，表示拓扑元素的边界的标识。

PostGIS 2.2.0dev 开发手册

542 / 686

**11.1.2**

**TopoGeometry**

TopoGeometry — 复合数据类型，用于定义几何类型对象的拓扑结构，可以称之为拓扑类型

**描述**

一种复合数据类型，它引用一个指定的拓扑图层里的拓扑几何对象，有一个具体的类型和一个具体的ID。一个TopoGeometry 类型包含的元素有： topology\_id, layer\_id, id integer, type integer.

1. topology\_id 是一个整型值。它引用定义在表topology.topology（该表定义拓扑的schema和SRID） 的一个拓扑结构。

2. layer\_id：一个整型值， layer\_id表示一个拓扑结合对象的某个图层。topology\_id, layer\_id 组合起来提供了在topology.layers 表中的唯一主键约束。

3. id：一个整型值，id是一个自动生成的序列 ，它唯一地定义一个拓扑图层中的拓扑几何对象。

4. type ：整型值，范围在1到4，用于定义几何对象类型： 1:[multi]point, 2:[multi]line, 3:[multi]poly, 4:collection

**转换方式**

这里列出的是自动转换方式，和该数据类型的显示转换一样。

**相关请参考**

CreateTopoGeom

**11.1.3**

**validatetopology\_returntype**

validatetopology\_returntype — 一种复合数据类型，包含错误信息和id1、id2两个用于表示错误信息位置的字段。是校验拓扑结构有效性函数ValidateTopology的返回值。

**描述**

一种复合数据类型，包含错误信息和id1、id2两个用于表示错误信息位置的字段。是校验拓扑结构有效性函数ValidateTopology的返回值。

1. error varchar类型，表示错误的类型

目前错误类型枚举如下：coincident nodes, edge crosses node, edge not simple, edge end node geometry mis-match,

edge start node geometry mismatch, face overlaps face,face within face,

2. id1 integer类型：表示错误信息中是哪个edge/face/node错了

3. id2 integer类型：对于包含两个以上对象的数据（层级数据），该字段表示第二个edge/node的错误位置

**相关请参考**

ValidateTopology

|  |  |
| --- | --- |
| 目标转换类型 | 转换方式 |
| geometry | 自动转换 |

PostGIS 2.2.0dev 开发手册

543 / 686

**11.2**

**11.2.1**

**拓扑域函数**

**TopoElement**

TopoElement —一个包含两个整型数字的数组，通常用于标识一个拓扑几何对象的组成部分。

**描述**

一个包含两个int型元素的数组，其中id（第一个数组元素）和type（第二个数组元素）表示一个拓扑的要素，或者id（第一个数组元素）和layer（第二个数组元素）表示一个TopoGeometry 拓扑结构。这样成对的集合用于定义TopoGeometry 对象（无论是简单的还是多层嵌套的）。

**样例**

SELECT te[1] AS id, te[2] AS type FROM

( SELECT ARRAY[1,2]::topology.topoelement AS te ) f;

id | type

----+------

1 | 2

SELECT ARRAY[1,2]::topology.topoelement;

te

-------

{1,2}

--样例 of what happens when you try to case a 3 element array to topoelement

-- NOTE: topoement has to be a 2 element array so fails dimension check

SELECT ARRAY[1,2,3]::topology.topoelement;

ERROR:

value for domain topology.topoelement violates check constraint "dimensions"

**相关请参考**

GetTopoGeomElements, TopoElementArray

**11.2.2**

**TopoElementArray**

TopoElementArray — 返回拓扑元素对象的数组

**描述**

一个包含1个或更多个TopoElement 类型对象的数组，一般用来传递TopoGeometry 对象的多个组成部分。

**样例**

SELECT ’{{1,2},{4,3}}’::topology.topoelementarray As tea;

tea

-------

{{1,2},{4,3}}

-- more verbose equivalent --

SELECT ARRAY[ARRAY[1,2], ARRAY[4,3]]::topology.topoelementarray As tea;

PostGIS 2.2.0dev 开发手册

544 / 686

tea

-------

{{1,2},{4,3}}

--using the array agg function packaged with topology --

SELECT topology.TopoElementArray\_Agg(ARRAY[e,t]) As tea

FROM generate\_series(1,4) As e CROSS JOIN generate\_series(1,3) As t;

tea

--------------------------------------------------------------------------

{{1,1},{1,2},{1,3},{2,1},{2,2},{2,3},{3,1},{3,2},{3,3},{4,1},{4,2},{4,3}}

SELECT ’{{1,2,4},{3,4,5}}’::topology.topoelementarray As tea;

ERROR:

value for domain topology.topoelementarray violates check constraint "dimensions"

**相关请参考**

TopoElement, GetTopoGeomElementArray, TopoElementArray\_Agg

**11.3**

**11.3.1**

**拓扑和拓扑几何对象管理**

**AddTopoGeometryColumn**

AddTopoGeometryColumn — 把一个topogeometry类型列添加到一个已经存在的表中，并且把这个列注册到系统表topology.layer中，然后返回layer\_id

**用法**

text AddTopoGeometryColumn(varchar topology\_name, varchar schema\_name, varchar table\_name, varchar column\_name,

varchar feature\_type);

text AddTopoGeometryColumn(varchar topology\_name, varchar schema\_name, varchar table\_name, varchar column\_name,

varchar feature\_type, integer child\_layer);

**描述**

每一个TopoGeometry对象属于一个指定拓扑结构的指定图层（layer）。在创建TopoGeometry类型对象之前，你需要先创建它的TopologyLayer（拓扑图层）。一个拓扑图层与它的拓扑特征表相关联，它也包含类型和隐含的信息。我们创建使用函数AddTopoGeometryColumn() 创建一个图层。这个函数会把列添加到表中，并且在系统表topology.layer中添加一条改列的记录信息。如果你不指定参数child\_layer (或者把它设置为NULL)，这个图层将会包含基本的TopoGeometries对象 (有原始的拓扑元素组成)，如果指定了这个参数那么这个图层将会包含嵌套的TopoGeometries (包含child\_layer对应的TopoGeometries)。一旦创建了图层（图层的ID由函数AddTopoGeometryColumn返回），你就已经做好了在图层里面创建好TopoGeometry对象的准备了

参数feature\_types的有效值有： POINT, LINE, POLYGON, COLLECTION

可用版本：1.?

PostGIS 2.2.0dev 开发手册

545 / 686

**样例**

-- 注意 for this example we created our new table in the ma\_topo schema

-- though we could have created it in a different schema -- in which case topology\_name and ←

schema\_name would be different

CREATE SCHEMA ma;

CREATE TABLE ma.parcels(gid serial, parcel\_id varchar(20) PRIMARY KEY, address text);

SELECT topology.AddTopoGeometryColumn(’ma\_topo’, ’ma’, ’parcels’, ’topo’, ’POLYGON’);

CREATE SCHEMA ri;

CREATE TABLE ri.roads(gid serial PRIMARY KEY, road\_name text);

SELECT topology.AddTopoGeometryColumn(’ri\_topo’, ’ri’, ’roads’, ’topo’, ’LINE’);

**相关请参考**

CreateTopology, CreateTopoGeom

**11.3.2**

**DropTopology**

DropTopology — 使用警告：这个函数会删除一个topology schema，并且删除该schema与系统表topology.topology和geometry\_columns之间的外键引用关系。

**用法**

integer DropTopology(varchar topology\_schema\_name);

**描述**

这个函数会删除一个topology schema，并且删除该schema与系统表topology.topology和geometry\_columns之间的外键引用关系。这个函数应该谨慎使用，因为它会损坏你关系的数据。如果这个schema不存在了，它只会删除使用该schema的引用入口。（译者注：也就是相关联的表不会删除的）

可用版本：1.?

**样例**

Cascade drops the ma\_topo schema and removes all references to it in topology.topology and geometry\_columns.

SELECT topology.DropTopology(’ma\_topo’);

**相关请参考**

**11.3.3**

**DropTopoGeometryColumn**

DropTopoGeometryColumn — 从schema为参数schema\_name表名为参数table\_name的表中删除一个topogeometry类型的列，并且把这个列的注册信息从系统表topology.layer 中删除。

**用法**

text DropTopoGeometryColumn(varchar schema\_name, varchar table\_name, varchar column\_name);

PostGIS 2.2.0dev 开发手册

546 / 686

**描述**

从schema为参数schema\_name表名为参数table\_name的表中删除一个topogeometry类型的列，并且把这个列的注册信息从系统表topology.layer 中删除，同时返回删除状态。注意：它首先将所有值设置为空，然后删除循环引用完整性约束

可用版本：1.?

**样例**

SELECT topology.DropTopoGeometryColumn(’ma\_topo’, ’parcel\_topo’, ’topo’);

**相关请参考**

AddTopoGeometryColumn

**11.3.4**

**TopologySummary**

TopologySummary — 根据一个topology name名称，提供一个topology对象里面所有对象类型的概括统计

**用法**

text TopologySummary(varchar topology\_schema\_name);

**描述**

根据一个topology name名称，提供一个topology对象里面所有对象类型的概括统计

可用版本：2.0.0

**样例**

SELECT topology.topologysummary(’city\_data’);

topologysummary

--------------------------------------------------------

Topology city\_data (329), SRID 4326, precision: 0

22 nodes, 24 edges, 10 faces, 29 topogeoms in 5 layers

Layer 1, type Polygonal (3), 9 topogeoms

Deploy: features.land\_parcels.feature

Layer 2, type Puntal (1), 8 topogeoms

Deploy: features.traffic\_signs.feature

Layer 3, type Lineal (2), 8 topogeoms

Deploy: features.city\_streets.feature

Layer 4, type Polygonal (3), 3 topogeoms

Hierarchy level 1, child layer 1

Deploy: features.big\_parcels.feature

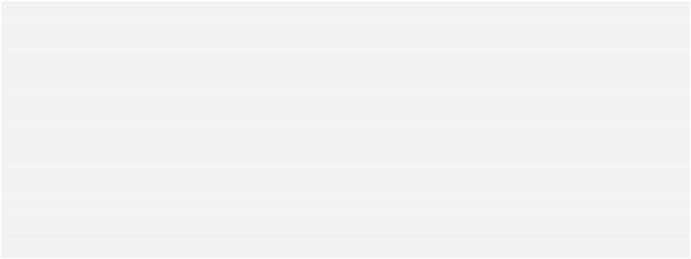
Layer 5, type Puntal (1), 1 topogeoms

Hierarchy level 1, child layer 2

Deploy: features.big\_signs.feature

**相关请参考**

Topology\_Load\_Tiger



PostGIS 2.2.0dev 开发手册

547 / 686

**11.3.5**

**ValidateTopology**

ValidateTopology — 返回类型为validatetopology\_returntype的记录集，用于对topology结构的有效性检查结果。

**用法**

setof validatetopology\_returntype ValidateTopology(varchar topology\_schema\_name);

**描述**

返回类型为validatetopology\_returntype的记录集，用于对topology结构的有效性检查结果。参考validatetopology\_returntype 获取所有可能的错误类别。

可用版本：1.?

版本提升： 2.0.0 更有效的边界交叉检测，修复了之前版本已经明确确定的若干问题。

**样例**

SELECT \* FROM

error

topology.ValidateTopology(’ma\_topo’);

| id1 | id2

-------------------+-----+-----

face without edges | 0 |

**相关请参考**

validatetopology\_returntype, Topology\_Load\_Tiger

**11.4**

**11.4.1**

**拓扑结构构造函数**

**CreateTopology**

CreateTopology — 创建一个拓扑结构的schema，并将这个schema注册到系统表the topology.topology 中

**用法**

integer CreateTopology(varchar topology\_schema\_name);

integer CreateTopology(varchar topology\_schema\_name, integer srid);

integer CreateTopology(varchar topology\_schema\_name, integer srid, double precision tolerance);

integer CreateTopology(varchar topology\_schema\_name, integer srid, double precision tolerance, boolean hasz);

**描述**

创建一个名为topology\_name的schema，该schema下面包含许多表(edge\_data（边界数据）,face（面）,node（节点）, relation（拓扑元素关系）)，并将这个schema注册到系统表topology.topology 中。它会返回topology.topology表中的拓扑ID。参数SRID是拓扑结构所在的空间参考系的SRID值，定义在系统表spatial\_ref\_sys table中。拓扑结构必须是命名唯一的。参数tolerance 测量单位以空间参考系的单位计算。如果没有指定参数tolerance，那么默认取值为0。

这个函数与SQL/MM规范的函数 ST\_InitTopoGeo 相似，但是有多一个功能。参数hasz如果没有指定，那么默认是false。

可用版本：1.?

PostGIS 2.2.0dev 开发手册

548 / 686

**样例**

This example creates a new schema called ma\_topo that will store edges, faces, and relations in Massachusetts State Plane meters.

The tolerance represents 1/2 meter since the spatial reference system is a meter based spatial reference system

SELECT topology.CreateTopology(’ma\_topo’,26986, 0.5);

Create Rhode Island topology in State Plane ft

SELECT topology.CreateTopology(’ri\_topo’,3438) As topoid;

topoid

------

2

**相关请参考**

Section 4.3.1, ST\_InitTopoGeo, Topology\_Load\_Tiger

**11.4.2**

**CopyTopology**

CopyTopology — 拷贝一个拓扑的结构(包括结点、边界、面、图层和拓扑结构).

**用法**

integer CopyTopology(varchar existing\_topology\_name, varchar new\_name);

**描述**

拷贝参数existing\_topology\_name 表示拓扑结构为一个新的名为new\_topology\_name的拓扑结构，两者SRID，精度相同，拷贝的内容包括，所有的结点、边界、面、图层和图层中的TopoGeometries对象

**注意**

系统表topology.layer的新记录会包含如下字段的“虚拟”值：schema\_name, table\_name and feature\_column。这是因为TopoGeometry对象只是进行了定义，还没有在任何用户实际的表中可用。

可用版本：2.0.0

**样例**

This example makes a backup of a topology called ma\_topo

SELECT topology.CopyTopology(’ma\_topo’, ’ma\_topo\_bakup’);

**相关请参考**

Section 4.3.1, CreateTopology

**11.4.3**

**ST\_InitTopoGeo**

ST\_InitTopoGeo —创建一个拓扑结构的schema，并将这个新schema注册到系统表the topology.topology 中，同时返回该函数处理过程的概要。



PostGIS 2.2.0dev 开发手册

549 / 686

**用法**

text ST\_InitTopoGeo(varchar topology\_schema\_name);

**描述**

这个函数是SQL-MM标准下的与函数CreateTopology等效的函数，但是这个函数缺少CreateTopology函数的空间参考SRID值和参数tolerance选项，并且输出输出的是创建过程描述，而不是拓扑结构的id\

可用版本：1.?

该方法实现了SQL-MM规范. SQL-MM 3 Topo-Geo and Topo-Net 3: Routine Details: X.3.17

**样例**

SELECT topology.ST\_InitTopoGeo(’topo\_schema\_to\_create’) AS topocreation;

astopocreation

------------------------------------------------------------

Topology-Geometry ’topo\_schema\_to\_create’ (id:7) created.

**相关请参考**

CreateTopology

**11.4.4**

**ST\_CreateTopoGeo**

ST\_CreateTopoGeo — 把一个几何对象的集合添加到一个指定的空拓扑结构中，然后返回处理成功的信息。

**用法**

text ST\_CreateTopoGeo(varchar atopology, geometry acollection);

**描述**

把一个几何对象的集合添加到一个指定的空拓扑结构中，然后返回处理成功的信息。

这个函数在填充拓扑结构中很有用。

可用版本：2.0

该方法实现了SQL-MM规范. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details -- X.3.18

**样例**

-- Populate topology --

SELECT topology.ST\_CreateTopoGeo(’ri\_topo’,

ST\_GeomFromText(’MULTILINESTRING((384744 236928,384750 236923,384769 236911,384799

←

236895,384811 236890,384833 236884,

384844 236882,384866 236881,384879 236883,384954 236898,385087 236932,385117 236938,

385167 236938,385203 236941,385224 236946,385233 236950,385241 236956,385254 236971,

385260 236979,385268 236999,385273 237018,385273 237037,385271 237047,385267 237057,

385225 237125,385210 237144,385192 237161,385167 237192,385162 237202,385159 237214,

385159 237227,385162 237241,385166 237256,385196 237324,385209 237345,385234 237375,



PostGIS 2.2.0dev 开发手册

550 / 686

385237 237383,385238 237399,385236

385174 237451,385170 237455,385169

385200 237533,385206 237538,385213

237407,385227

237460,385171

237541,385221

237419,385213

237475,385181

237542,385235

237430,385193 237439,

237503,385190 237521,

237540,385242 237541,

385249 237544,385260 237555,385270

237570,385289

237584,385292

237589,385291

←

237596,385284 237630))’,3438)

);

st\_createtopogeo

----------------------------

Topology ri\_topo populated

-- create tables and topo geometries --

CREATE TABLE ri.roads(gid serial PRIMARY KEY, road\_name text);

SELECT topology.AddTopoGeometryColumn(’ri\_topo’, ’ri’, ’roads’, ’topo’, ’LINE’);

**相关请参考**

AddTopoGeometryColumn, CreateTopology, DropTopology

**11.4.5**

**TopoGeo\_AddPoint**

TopoGeo\_AddPoint —把一个点几何对象添加到一个已经存在的拓扑结构中，点添加的位置可能会把一个已经存在的edge边界分割开。

**用法**

integer TopoGeo\_AddPoint(varchar toponame, geometry apoint, ﬂoat8 tolerance);

**描述**

把一个点几何对象添加到一个已经存在的拓扑结构中，返回点的ID。在给定的tolerance容差内，给定的点会与已经存在的结点或边界保持对齐（可能在一条线上）。一个已经存在的边界可能会被新增的点分隔开。

可用版本：2.0.0

**相关请参考**

TopoGeo\_AddLineString, TopoGeo\_AddPolygon, AddNode, CreateTopology

**11.4.6**

**TopoGeo\_AddLineString**

TopoGeo\_AddLineString —把一个linestring几何对象在指定参数tolerance容差下，添加到一个已经存在的拓扑结构中，linestring添加的位置可能会把一个已经存在的edge边界或面分割开。函数会返回edge边界的ID

**用法**

SETOF integer TopoGeo\_AddLineString(varchar toponame, geometry aline, ﬂoat8 tolerance);

**描述**

把一个linestring几何对象在指定参数tolerance容差下，添加到一个已经存在的拓扑结构中，linestring添加的位置可能会把一个已经存在的edge边界或面分割开。函数会返回edge边界的ID

可用版本：2.0.0

PostGIS 2.2.0dev 开发手册

551 / 686

**相关请参考**

TopoGeo\_AddPoint, TopoGeo\_AddPolygon, AddEdge, CreateTopology

**11.4.7**

**TopoGeo\_AddPolygon**

TopoGeo\_AddPolygon —把一个polygon几何对象在指定参数tolerance容差下，添加到一个已经存在的拓扑结构中，polygon几何对象添加的位置可能会把一个已经存在的edge边界或面分割开。

**用法**

integer TopoGeo\_AddPolygon(varchar atopology, geometry apoly, ﬂoat8 atolerance);

**描述**

把一个polygon几何对象在指定参数tolerance容差下，添加到一个已经存在的拓扑结构中，并且返回面ID的集合。polygon几何对象添加的位置可能会把一个已经存在的edge边界或面分割开。

可用版本：2.0.0

**相关请参考**

TopoGeo\_AddPoint, TopoGeo\_AddLineString, AddFace, CreateTopology

**11.5**

**11.5.1**

**拓扑结构编辑函数**

**ST\_AddIsoNode**

ST\_AddIsoNode — 在一个拓扑结构的一个面上创建一个孤立的结点并返回新的结点的id。如果面是null空值，那么结点依然会被创建

**用法**

integer ST\_AddIsoNode(varchar atopology, integer aface, geometry apoint);

**描述**

根据提供表示点位置的参数apoint，在已经存在的拓扑结构atopology的指定aface的face面对象中添加一个孤立点，然后返回新结点的结点ID。如果添加的point对象的SRID和拓扑结构的SRID值不同，或者apoint不是点几何对象，或者point是null，或者这个点已经与已经存在的边相交，那么都会抛出异常。如果点已经作为一个结点存在，那么也会抛出异常。如果参数aface不是null的，apoint也不在面内，同样也会抛出异常。

可用版本：1.?

该方法实现了SQL-MM规范. SQL-MM: Topo-Net Routines: X+1.3.1



PostGIS 2.2.0dev 开发手册

552 / 686

**样例**

**相关请参考**

AddNode, CreateTopology, DropTopology, ST\_Intersects

**11.5.2**

**ST\_AddIsoEdge**

ST\_AddIsoEdge — 添加一个由参数alinestring定义的孤立的边到一个拓扑结构中，用于连接两个已经存在的由参数anode和anothernode定义的孤立点。然后返回新边的边ID。

**用法**

integer ST\_AddIsoEdge(varchar atopology, integer anode, integer anothernode, geometry alinestring);

**描述**

添加一个由参数alinestring定义的孤立的边到一个拓扑结构中，用于连接两个已经存在的由参数anode和anothernode定义的孤立点。然后返回新边的边ID。

如果添加的alinestring对象的SRID和拓扑结构的SRID值不同，或者任意输入参数是null，或者这个alinestring的结点被不止一个面包含，或者alinestring的起始和终结结点已经与已经存在的边相交，那么都会抛出异常。

如果点已经作为一个结点存在，那么也会抛出异常。如果参数aface不是null的，apoint也不在面内，同样也会抛出异常。

如果alinestring不在参数anode和anothernode所属的面内，那么会抛出异常。

如果参数anode和anothernode不是alinestring的起点和终点，那么会抛出异常。

可用版本：1.?

该方法实现了SQL-MM规范. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.4

**样例**

**相关请参考**

ST\_AddIsoNode, ST\_IsSimple, ST\_Within

**11.5.3**

**ST\_AddEdgeNewFaces**

ST\_AddEdgeNewFaces —在一个face面内添加一条新边界，把这个face面分开，然后删除掉原始的面，并用两个新的面来替换。

**用法**

integer ST\_AddEdgeNewFaces(varchar atopology, integer anode, integer anothernode, geometry acurve);



PostGIS 2.2.0dev 开发手册

553 / 686

**描述**

在一个face面内添加一条新边界，把这个face面分开，然后删除掉原始的面，并用两个新的面来替换。

相应地更新已有的相关联的边和边的关系。

如果任意输入参数是null，或者函数的参数结点未知（这个结点必须是topology schema中的结点表中的数据），或者参数acurve不是一个LINESTRING对象，或者参数anode和anothernode不是alinestring的起点和终点，那么都会抛出异常。

如果添加的acurve对象的SRID和拓扑结构的SRID值不同，那么也会抛出异常。

可用版本：2.0

该方法实现了SQL-MM规范. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.12

**样例**

**相关请参考**

ST\_RemEdgeNewFace

ST\_AddEdgeModFace

**11.5.4**

**ST\_AddEdgeModFace**

ST\_AddEdgeModFace —在一个face面内添加一条新的边，把这个face面分开，然后修改原始的面并添加一个新的面。

**用法**

integer ST\_AddEdgeModFace(varchar atopology, integer anode, integer anothernode, geometry acurve);

**描述**

在一个face面内添加一条新的边，把这个face面分开，然后修改原始的面并添加一个新的面。除非被分开的面是Universal Face（超大的宇宙面），否则新的面将会在新添加的边的右侧。该函数会返回新添加边的edgeID。

相应地更新已有的相关联的边和边的关系。

如果任意输入参数是null，或者函数的参数结点未知（这个结点必须是topology schema中的结点表中的数据），或者参数acurve不是一个LINESTRING对象，或者参数anode和anothernode不是alinestring的起点和终点，那么都会抛出异常。

如果添加的acurve对象的SRID和拓扑结构的SRID值不同，那么也会抛出异常。

可用版本：2.0

该方法实现了SQL-MM规范. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.13

**样例**

**相关请参考**

ST\_RemEdgeModFace

ST\_AddEdgeNewFaces



PostGIS 2.2.0dev 开发手册

554 / 686

**11.5.5**

**ST\_RemEdgeNewFace**

ST\_RemEdgeNewFace — 移除一个边，如果移除的边分隔了两个面，删除原始的面并使用一个新的面进行替换。

**用法**

integer ST\_RemEdgeNewFace(varchar atopology, integer anedge);

**描述**

移除一个边，如果移除的边分隔了两个面，删除原始的面并使用一个新的面进行替换。

返回新创建的face面ID，如果没有创建新的面，那么返回为NULL。当被移除的面是孤立的或者局限于宇宙面（可能使宇宙面涌入另一面），那么将不会创建新的面。

相应地更新已有的相关联的边和边的关系。

该函数不会移除一个参与定义了已经存在的TopoGeometry的边

如果任意一个TopoGeometry 对象是由两个面的其中之一定义，那么将不会更新着两个面。

如果任意输入参数是null，或者函数的参数结点未知（这个结点必须是topology schema中的结点表中的数据），或者参数atopology 无效，那么都会抛出错误。

可用版本：2.0

该方法实现了SQL-MM规范. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.14

**样例**

**相关请参考**

ST\_RemEdgeModFace

ST\_AddEdgeNewFaces

**11.5.6**

**ST\_RemEdgeModFace**

ST\_RemEdgeModFace —移除一个边，如果移除的边分隔了两个面，删除其中面的其中之一，然后修改另一个，让它占领原本属于两个面的空间，即占有原始的未分开的面。

**用法**

integer ST\_RemEdgeModFace(varchar atopology, integer anedge);

**描述**

移除一个边，如果移除的边分隔了两个面，删除其中面的其中之一，然后修改另一个，让它占领原本属于两个面的空间，即占有原始的未分开的面。优先保留边右边的面，与ST\_AddEdgeModFace保持对称的面也会保留。返回移除一个边后依然保留的面的ID

相应地更新已有的相关联的边和边的关系。

该函数不会移除一个参与定义了已经存在的TopoGeometry的边

如果任意一个TopoGeometry 对象是由两个面的其中之一定义，那么将不会更新着两个面。



PostGIS 2.2.0dev 开发手册

555 / 686

如果任意输入参数是null，或者函数的参数结点未知（这个结点必须是topology schema中的结点表中的数据），或者参数atopology 无效，那么都会抛出错误。

可用版本：2.0

该方法实现了SQL-MM规范. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.15

**样例**

**相关请参考**

ST\_AddEdgeModFace

ST\_RemEdgeNewFace

**11.5.7**

**ST\_ChangeEdgeGeom**

ST\_ChangeEdgeGeom — 不改变拓扑结构的情况下，改变一个边的形状。

**用法**

integer ST\_ChangeEdgeGeom(varchar atopology, integer anedge, geometry acurve);

**描述**

不改变拓扑结构的情况下，改变一个边的形状。

如果任意输入参数是null，或者函数的参数结点未知（这个结点必须是topology schema中的结点表中的数据），或者参数acurve不是一个LINESTRING对象，或者参数anode和anothernode不是alinestring的起点和终点，或者修改会改变底层的拓扑结构，那么都会抛出一个错误。

如果添加的acurve对象的SRID和拓扑结构的SRID值不同，那么也会抛出一个错误。

如果参数acurve不是简单的曲线，那么将会抛出一个错误。

如果把边从旧的位置移动到一个新的位置时候遇到障碍，那么将会抛出一个错误。

可用版本：1.1.0

版本提升： 2.0.0 添加了拓扑一致性的强制校验

该方法实现了SQL-MM规范. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details X.3.6

**样例**

SELECT topology.ST\_ChangeEdgeGeom(’ma\_topo’, 1,

ST\_GeomFromText(’LINESTRING(227591.9 893900.4,227622.6 893844.3,227641.6 893816.6,

227704.5 893778.5)’, 26986) );

----

Edge 1 changed

**相关请参考**

ST\_AddEdgeModFace

ST\_RemEdgeModFace

ST\_ModEdgeSplit

←



PostGIS 2.2.0dev 开发手册

556 / 686

**11.5.8**

**ST\_ModEdgeSplit**

ST\_ModEdgeSplit —在一个已经存在的边上通过创建一个结点来分割一个边，这样就修改了原始的边并添加了一个新的边（带有结点）。

**用法**

integer ST\_ModEdgeSplit(varchar atopology, integer anedge, geometry apoint);

**描述**

在一个已经存在的边上通过创建一个结点来分割一个边，这样就修改了原始的边并添加了一个新的边（带有结点）。

相应地更新所有和原有边有关联的边以及边的关系。

该函数会返回新添加的结点的ID。

可用版本：1.?

版本变更：2.0 –在之前的版本，这个函数名被误定义为ST\_ModEdgesSplit

该方法实现了SQL-MM规范. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.9

**样例**

-- Add an edge --

SELECT topology.AddEdge(’ma\_topo’, ST\_GeomFromText(’LINESTRING(227592 893910, 227600

893910)’, 26986) ) As edgeid;

-- edgeid-

3

←

-- Split the edge

--

SELECT topology.ST\_ModEdgeSplit(’ma\_topo’,

3, ST\_SetSRID(ST\_Point(227594,893910),26986)

) ←

As node\_id;

node\_id

-------------------------

7

**相关请参考**

ST\_NewEdgesSplit, ST\_ModEdgeHeal, ST\_NewEdgeHeal, AddEdge

**11.5.9**

**ST\_ModEdgeHeal**

ST\_ModEdgeHeal — 解开两个边的连接，通过删除两个边之间的结点，然后更正第一个边的信息（就是第一个边不再带有结点了），再把第二个边删掉。这个函数返回删除结点的ID。

**用法**

int ST\_ModEdgeHeal(varchar atopology, integer anedge, integer anotheredge);



PostGIS 2.2.0dev 开发手册

557 / 686

**描述**

解开两个边的连接，通过删除两个边之间的结点，然后更正第一个边的信息（就是第一个边不再带有结点了），再把第二个边删掉。这个函数返回删除结点的ID。

相应地更新所有和原有边有关联的边以及边的关系。

可用版本：2.0

该方法实现了SQL-MM规范. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.9

**相关请参考**

ST\_ModEdgeSplit ST\_NewEdgesSplit

**11.5.10**

**ST\_NewEdgeHeal**

ST\_NewEdgeHeal — 修复两个边界：先删除连接两个边界的结点，然后删除两个边界，然后再用与第一个边界anedge方向相同的边界替换这两个边界。

**用法**

int ST\_NewEdgeHeal(varchar atopology, integer anedge, integer anotheredge);

**描述**

修复两个边界：先删除连接两个边界的结点，然后删除两个边界，然后再用与第一个边界anedge方向相同的边界替换这两个边界。返回替换修复过原始边界的新边界ID。

相应地更新所有和原有边有关联的边以及边的关系。

可用版本：2.0

该方法实现了SQL-MM规范. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.9

**相关请参考**

ST\_ModEdgeHeal ST\_ModEdgeSplit ST\_NewEdgesSplit

**11.5.11**

**ST\_MoveIsoNode**

ST\_MoveIsoNode —把拓扑结构内的一个孤立点从一个位置移动到另一个位置，如果目标位置是一个已经存在的节点，则抛出异常。返回该移动操作的详细信息。

**用法**

text ST\_MoveIsoNode(varchar atopology, integer anedge, geometry apoint);



PostGIS 2.2.0dev 开发手册

558 / 686

**描述**

把拓扑结构内的一个孤立点从一个位置移动到另一个位置，如果目标位置是一个已经存在的节点，则抛出异常。返回该移动操作的详细信息。

如果函数的任意参数是null，或者参数apoint不是一个点，或者已经存在的结点不是孤立的（比如是一个已经存在的边的起点或者终点），或者新的结点位置与已经存在的边相交（即便是在终点相交），那么都会抛出一个错误。

如果点几何对象的SRID和拓扑的SRID值不一致，也都会抛出错误。

可用版本：1.?

该方法实现了SQL-MM规范. SQL-MM: Topo-Net Routines: X.3.2

**样例**

-- Add an isolated node with no face

--

SELECT topology.ST\_AddIsoNode(’ma\_topo’,

26986) ) As nodeid;

nodeid

--------

7

-- Move the new node --

NULL, ST\_GeomFromText(’POINT(227579 893916)’,

←

SELECT topology.ST\_MoveIsoNode(’ma\_topo’, 7,

ST\_GeomFromText(’POINT(227579.5 893916.5)’,

←

26986) ) As descrip;

descrip

----------------------------------------------------

Isolated Node 7 moved to location 227579.5,893916.5

**相关请参考**

ST\_AddIsoNode

**11.5.12**

**ST\_NewEdgesSplit**

ST\_NewEdgesSplit —分割一个边：首先在一个已经存在的边上创建一个新的结点，然后删除原有的边界，然后用两个新的边界替换原始边界。这个函数返回连接两个边界的结点。

**用法**

integer ST\_NewEdgesSplit(varchar atopology, integer anedge, geometry apoint);

**描述**

分割一个边：首先在一个已经存在的边上创建一个新的结点，然后删除原有的边界，然后用两个新的边界替换原始边界。这个函数返回连接两个边界的结点。

相应地更新所有和原有边有关联的边以及边的关系。

如果点几何对象的SRID和拓扑的SRID值不一致，或者参数apoint不是一个点，或者参数point是null，或者点是已经存在的结点，或者输入参数anedge不是一个已经存在的边，或者point不在edge边上面，那么都会抛出一个错误。

可用版本：1.?

该方法实现了SQL-MM规范. SQL-MM: Topo-Net Routines: X.3.8



PostGIS 2.2.0dev 开发手册

559 / 686

**样例**

-- Add an edge

--

SELECT topology.AddEdge(’ma\_topo’, ST\_GeomFromText(’LINESTRING(227575 893917,227592 893900) ←

’, 26986) ) As edgeid;

-- result-

edgeid

------

2

-- Split the new edge --

SELECT topology.ST\_NewEdgesSplit(’ma\_topo’, 2,

ST\_GeomFromText(’POINT(227578.5 893913.5)’, ←

26986) ) As newnodeid;

newnodeid

---------

6

**相关请参考**

ST\_ModEdgeSplit ST\_ModEdgeHeal ST\_NewEdgeHeal AddEdge

**11.5.13**

**ST\_RemoveIsoNode**

ST\_RemoveIsoNode — 删除孤立点并返回该操作的描述信息。如果点不上孤立点（比如是edge（边）起始或者终点），那么会抛出异常。

**用法**

text ST\_RemoveIsoNode(varchar atopology, integer anode);

**描述**

删除孤立点并返回该操作的描述信息。如果点不上孤立点（比如是edge（边）起始或者终点），那么会抛出异常。

可用版本：1.?

该方法实现了SQL-MM规范. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X+1.3.3

**样例**

-- Add an isolated node with no face

--

SELECT topology.ST\_RemoveIsoNode(’ma\_topo’,

result

-------------------------

Isolated node 7 removed

**相关请参考**

ST\_AddIsoNode

7 ) As result;



PostGIS 2.2.0dev 开发手册

560 / 686

**11.6**

**11.6.1**

**拓扑结构存取函数**

**GetEdgeByPoint**

GetEdgeByPoint — 根据给定的点，返回和该点相交的边界ID

**用法**

integer GetEdgeByPoint(varchar atopology, geometry apoint, ﬂoat8 tol);

根据给定的点，返回和该点相交的边界ID

根据输入的3个参数，该函数返回一个edge的ID 整型值。如果参数tolerance=0，那么点必须要与edge相交。如果输入的点是节点的位置，那么会抛出异常。为了避免这种情况，运行函数GetNodeByPoint。

如果点和输入的edge不相交，则返回0

如果输入参数tolerance>0，那么会有多个edge靠近该点，这样会抛出异常

**注意**

如果tolerance = 0，该函数使用函数ST\_Intersects，否则调用ST\_DWithin.

可用版本：2.0.0 - requires GEOS >= 3.3.0.

**样例**

These examples use edges we created in AddEdge

SELECT topology.GetEdgeByPoint(’ma\_topo’,geom, 1) As with1mtol, topology.GetEdgeByPoint(’ ←

ma\_topo’,geom,0) As withnotol

FROM ST\_GeomFromEWKT(’SRID=26986;POINT(227622.6 893843)’) As geom;

with1mtol | withnotol

-----------+-----------

2 | 0

SELECT topology.GetEdgeByPoint(’ma\_topo’,geom, 1) As nearnode

FROM ST\_GeomFromEWKT(’SRID=26986;POINT(227591.9 893900.4)’) As geom;

-- get error --

ERROR:

Two or more edges found

**相关请参考**

AddEdge, GetNodeByPoint

**11.6.2**

**GetFaceByPoint**

GetFaceByPoint — 根据给定的点，查找与该点相交的面id值。



PostGIS 2.2.0dev 开发手册

561 / 686

**用法**

integer GetFaceByPoint(varchar atopology, geometry apoint, ﬂoat8 tol);

**描述**

根据给定的点，查找与该点相交的面id值。

根据输入的3个参数，该函数返回一个面ID 整型值。如果参数tolerance=0，那么点必须要与face相交。如果输入的点是节点的位置，那么会抛出异常。为了避免这种情况，运行函数GetNodeByPoint。

如果点和输入的面不相交，则返回0

如果输入参数tolerance>0，那么会有多个面靠近该点，这样会抛出异常

**注意**

如果tolerance = 0，该函数使用函数ST\_Intersects，否则调用ST\_DWithin.

可用版本：2.0.0 - requires GEOS >= 3.3.0.

**样例**

These examples use edges faces created in AddFace

SELECT topology.GetFaceByPoint(’ma\_topo’,geom, 10) As with1mtol, topology.GetFaceByPoint(’ ←

ma\_topo’,geom,0) As withnotol

FROM ST\_GeomFromEWKT(’POINT(234604.6 899382.0)’) As geom;

with1mtol | withnotol

-----------+-----------

1 | 0

SELECT topology.GetFaceByPoint(’ma\_topo’,geom, 1) As nearnode

FROM ST\_GeomFromEWKT(’POINT(227591.9 893900.4)’) As geom;

-- get error --

ERROR:

Two or more faces found

**相关请参考**

AddFace, GetNodeByPoint, GetEdgeByPoint

**11.6.3**

**GetNodeByPoint**

GetNodeByPoint — 找出一个点位置的结点的ID。

**用法**

integer GetNodeByPoint(varchar atopology, geometry point, ﬂoat8 tol);



PostGIS 2.2.0dev 开发手册

562 / 686

找出一个点位置的结点的ID。

这个函数根据参数topology，point和tolerance（容差）返回一个整型值（结点的ID）。如果tolerance = 0 意味着完全位置完全重合，否则将从一个区间获取结点。如果参数point点位置上没有结点，那么函数将返回0。如果使用tolerance>0 ，并且靠近point多于一个结点，那么函数将会抛出一个异常。

**注意**

如果tolerance = 0，该函数使用函数ST\_Intersects，否则调用ST\_DWithin..

可用版本：2.0.0 –需要 GEOS >= 3.3.0.

**样例**

These examples use edges we created in AddEdge

SELECT topology.GetNodeByPoint(’ma\_topo’,geom, 1) As nearnode

FROM ST\_GeomFromEWKT(’SRID=26986;POINT(227591.9 893900.4)’) As geom;

nearnode

----------

2

SELECT topology.GetNodeByPoint(’ma\_topo’,geom, 1000) As too\_much\_tolerance

FROM ST\_GeomFromEWKT(’SRID=26986;POINT(227591.9 893900.4)’) As geom;

----get error--

ERROR:

Two or more nodes found

**相关请参考**

AddEdge, GetEdgeByPoint

**11.6.4**

**GetTopologyID**

GetTopologyID —根据拓扑名称，返回一个在topology.topology 拓扑表中的拓扑结构的ID值

**用法**

integer GetTopologyID(varchar toponame);

**描述**

根据拓扑名称，返回一个在topology.topology 拓扑表中的拓扑结构的ID值

可用版本：1.?

**样例**

SELECT topology.GetTopologyID(’ma\_topo’) As topo\_id;

topo\_id

---------

1



PostGIS 2.2.0dev 开发手册

563 / 686

**相关请参考**

CreateTopology, DropTopology, GetTopologyName, GetTopologySRID

**11.6.5**

**GetTopologySRID**

GetTopologySRID —根据拓扑名称，返回一个在topology.topology 拓扑表中的拓扑结构的SRID值

**用法**

integer GetTopologyID(varchar toponame);

**描述**

根据拓扑名称，返回一个在topology.topology 拓扑表中的拓扑结构的SRID值

可用版本：2.0.0

**样例**

SELECT topology.GetTopologySRID(’ma\_topo’) As SRID;

SRID

-------

4326

**相关请参考**

CreateTopology, DropTopology, GetTopologyName, GetTopologyID

**11.6.6**

**GetTopologyName**

GetTopologyName — 根据拓扑编号，返回一个拓扑结构的拓扑名称。

**用法**

varchar GetTopologyName(integer topology\_id);

**描述**

根据拓扑编号，返回一个拓扑结构的拓扑名称。根据系统表topology.topology的拓扑ID值，返回一个拓扑结构的拓扑名称

可用版本：1.?

**样例**

SELECT topology.GetTopologyName(1) As topo\_name;

topo\_name

-----------

ma\_topo

PostGIS 2.2.0dev 开发手册

564 / 686

**相关请参考**

CreateTopology, DropTopology, GetTopologyID, GetTopologySRID

**11.6.7**

**ST\_GetFaceEdges**

ST\_GetFaceEdges — 返回包围一个面的有序边界集合

**用法**

getfaceedges\_returntype ST\_GetFaceEdges(varchar atopology, integer aface);

**描述**

返回包围一个面的有序边界集合。每一行输出包含了一个序列和edgeid（边id）。序列号从1开始。每一个环的边从边id最小的开始计数。边的顺序按照左手定则（面必须在每一个有向边的左侧）来确定的。

可用版本：2.0

该方法实现了SQL-MM规范. SQL-MM 3 Topo-Geo and Topo-Net 3: Routine Details: X.3.5

**样例**

-- Returns the edges bounding face 1

SELECT (topology.ST\_GetFaceEdges(’tt’, 1)).\*;

-- result --

sequence | edge

----------+------

1 | -4

2 | 5

3 | 7

4 | -6

5 | 1

6 | 2

7 | 3

(7 rows)

-- Returns the sequenc, edge id

-- , and geometry of the edges that bound face 1

-- If you just need geom and seq, can use ST\_GetFaceGeometry

SELECT t.seq, t.edge, geom

FROM topology.ST\_GetFaceEdges(’tt’,1) As t(seq,edge)

INNER JOIN tt.edge AS e ON abs(t.edge) = e.edge\_id;

**相关请参考**

GetRingEdges, AddFace, ST\_GetFaceGeometry

**11.6.8**

**ST\_GetFaceGeometry**

ST\_GetFaceGeometry —根据给定的拓扑结构和面ID，返回一个polygon 对象



PostGIS 2.2.0dev 开发手册

565 / 686

**用法**

geometry ST\_GetFaceGeometry(varchar atopology, integer aface);

**描述**

根据给定的拓扑结构和面ID，返回一个polygon 对象。该polygon对象就是组成面的边界。

可用版本：1.?

该方法实现了SQL-MM规范. SQL-MM 3 Topo-Geo and Topo-Net 3: Routine Details: X.3.16

**样例**

-- Returns the wkt of the polygon added with AddFace

SELECT ST\_AsText(topology.ST\_GetFaceGeometry(’ma\_topo’, 1)) As facegeomwkt;

-- result --

facegeomwkt

--------------------------------------------------------------------------------

POLYGON((234776.9 899563.7,234896.5 899456.7,234914 899436.4,234946.6 899356.9,

234872.5 899328.7,234891 899285.4,234992.5 899145,234890.6 899069,

234755.2 899255.4,234612.7 899379.4,234776.9 899563.7))

**相关请参考**

AddFace

**11.6.9**

**GetRingEdges**

GetRingEdges — 根据给定的边返回一个环的有序边界集合。

**用法**

getfaceedges\_returntype GetRingEdges(varchar atopology, integer aring, integer max\_edges=null);

**描述**

根据给定的边，返回一个组成环的有序边集合。每一行输出包含了一个序列和一个有符号的边ID。序列号从1开始。如果边ID值是负数，表示给定的边是反向的。你可以传递一个负的edgeid表示从反向开始输出。如果参数max\_edges值不是null，那么函数返回的记录数将不超过参数max\_edges的值。当可能遇到无效的拓扑结构时候，这是一个安全的参数。

可用版本：2.0

**相关请参考**

ST\_GetFaceEdges, GetNodeEdges



PostGIS 2.2.0dev 开发手册

566 / 686

**11.6.10**

**GetNodeEdges**

GetNodeEdges — 根据给定的结点，跟着该结点的有序边。

**用法**

getfaceedges\_returntype GetNodeEdges(varchar atopology, integer anode);

**描述**

根据给定的结点，跟着该结点的有序边。每一行输出包含了一个序列和一个有符号的边ID。序列号从1开始。一个正向的边的起点是给定的结点，一个负向的边终点是给定的点。封闭的边会出现两次（边id都是有符号的数字）。顺序是从北向开始顺时针开始计算。

Closed edges will appear twice (with both signs). Order is clockwise starting from northbound.

**注意**

这个函数计算边的顺序而不是从元数据中读取数据，因此能用于建立循环边。

可用版本：2.0

**相关请参考**

GetRingEdges, ST\_Azimuth

**11.7**

**11.7.1**

**拓扑处理函数**

**Polygonize 多边形化**

Polygonize — 找出并注册所有由拓扑边定义的面。

**用法**

text Polygonize(varchar toponame);

**描述**

找出并注册所有由拓扑边定义的面。目标拓扑结构假定包含了没有自相交的边。

**注意**

Already known faces are recognized, so it is safe to call Polygonize multiple times on the same topology.

**注意**

This function does not use nor set the next\_left\_edge and next\_right\_edge ﬁelds of the edge table.

可用版本：2.0.0



PostGIS 2.2.0dev 开发手册

567 / 686

**相关请参考**

AddFace, ST\_Polygonize

**11.7.2**

**AddNode**

AddNode — Adds a point node to the node table in the speciﬁed topology schema and returns the nodeid of new node. If point

already exists as node, the existing nodeid is returned.

**用法**

integer AddNode(varchar toponame, geometry apoint, boolean allowEdgeSplitting=false, boolean computeContainingFace=false);

**描述**

Adds a point node to the node table in the speciﬁed topology schema. The AddEdge function automatically adds start and end

points of an edge when called so not necessary to explicitly add nodes of an edge.

If any edge crossing the node is found either an exception is raised or the edge is splitted, depending on the allowEdgeSpl-

itting parameter value.

If computeContainingFace is true a newly added node would get the correct containing face computed.

**注意**

If the apoint geometry already exists as a node, the node is not added but the existing nodeid is returned.

可用版本：2.0.0

**样例**

SELECT topology.AddNode(’ma\_topo’, ST\_GeomFromText(’POINT(227641.6 893816.5)’, 26986) ) As

nodeid;

-- result --

nodeid

--------

4

**相关请参考**

AddEdge, CreateTopology

←

**11.7.3**

**AddEdge**

AddEdge — Adds a linestring edge to the edge table and associated start and end points to the point nodes table of the speciﬁed

topology schema using the speciﬁed linestring geometry and returns the edgeid of the new (or existing) edge.

**用法**

integer AddEdge(varchar toponame, geometry aline);



PostGIS 2.2.0dev 开发手册

568 / 686

**描述**

Adds an edge to the edge table and associated nodes to the nodes table of the speciﬁed toponame schema using the speciﬁed

linestring geometry and returns the edgeid of the new or existing record. The newly added edge has "universe" face on both sides

and links to itself.

**注意**

If the aline geometry crosses, overlaps, contains or is contained by an existing linestring edge, then an error is thrown

and the edge is not added.

**注意**

The geometry of aline must have the same srid as deﬁned for the topology otherwise an invalid spatial reference

sys error will be thrown.

可用版本：2.0.0 requires GEOS >= 3.3.0.

**样例**

SELECT topology.AddEdge(’ma\_topo’, ST\_GeomFromText(’LINESTRING(227575.8 893917.2,227591.9

893900.4)’, 26986) ) As edgeid;

-- result-

edgeid

--------

1

SELECT topology.AddEdge(’ma\_topo’, ST\_GeomFromText(’LINESTRING(227591.9 893900.4,227622.6

893844.2,227641.6 893816.5,

227704.5 893778.5)’, 26986) ) As edgeid;

-- result --

edgeid

--------

2

SELECT topology.AddEdge(’ma\_topo’, ST\_GeomFromText(’LINESTRING(227591.2 893900, 227591.9

893900.4,

227704.5 893778.5)’, 26986) ) As edgeid;

-- gives error --

←

←

←

ERROR:

Edge intersects (not on endpoints) with existing edge 1

**相关请参考**

CreateTopology, Section 4.3.1

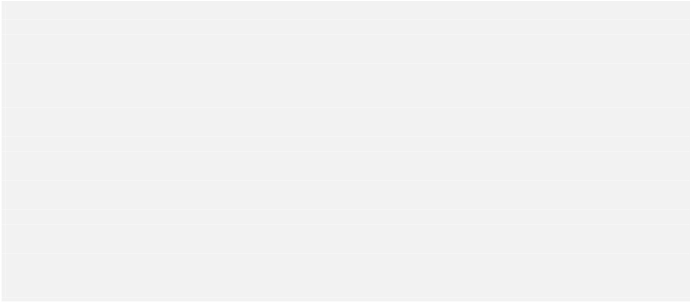
**11.7.4**

**AddFace**

AddFace — 把一个面要素添加到一个拓扑结构中，然后获取它的面ID。

**用法**

integer AddFace(varchar toponame, geometry apolygon, boolean force\_new=false);



PostGIS 2.2.0dev 开发手册

569 / 686

**描述**

把一个面要素添加到一个拓扑结构中，然后获取它的面ID。对于新添加的面，组成该面的边和面里面的对象会被更新成由参数left\_face和参数right\_face设置的值。面中的孤立点会被更新成参数containing\_face的值。

**注意**

这个函数不使用也不设置edge表的参数next\_left\_edge 和 next\_right\_edge。

函数参数topologyname被假定是有效的（不包含自相交的边）。如果出现下面这些情况那么将抛出异常：如果多边形边界不是由已经存在的边完全定义或多边形与一个已经存在的面交叠。

如果参数apolygon几何对象已经作为一个面存在，那么如果参数force\_new设置为false (即函数的默认值)，那么返回的是已经存在的face面的id；如果force\_new 设置为true，那么返回的是新生成的面的ID。

**注意**

当一个已经存在的面被注册（即添加到系统表中）时候（force\_new=true），那么函数不会做任何操作去处理一个引用不定的已经存在的面的边，节点等对象，同时已经存在的面记录的MBR字段也不会更新。由调用者决定怎样处理。

**注意**

参数apolygon多边形几何对象必须和拓扑结构的SRID值一致，否则将会抛出一个无效的空间参考系错误。

可用版本：2.0.0

**样例**

-- first add the edges we use generate\_series as an iterator (the below

-- will only work for polygons with < 10000 points because of our max in gs)

SELECT topology.AddEdge(’ma\_topo’, ST\_MakeLine(ST\_PointN(geom,i), ST\_PointN(geom, i + 1) )) ←

As edgeid

FROM (SELECT

FROM

ST\_NPoints(geom) AS npt, geom

(SELECT ST\_Boundary(ST\_GeomFromText(’POLYGON((234896.5 899456.7,234914

899436.4,234946.6 899356.9,234872.5 899328.7,

234891 899285.4,234992.5 899145, 234890.6 899069,234755.2 899255.4,

←

234612.7 899379.4,234776.9 899563.7,234896.5 899456.7))’, 26986) )

As geom

)

As geoms) As facen CROSS JOIN generate\_series(1,10000) As i

WHERE i < npt;

-- result --

edgeid

--------

3

4

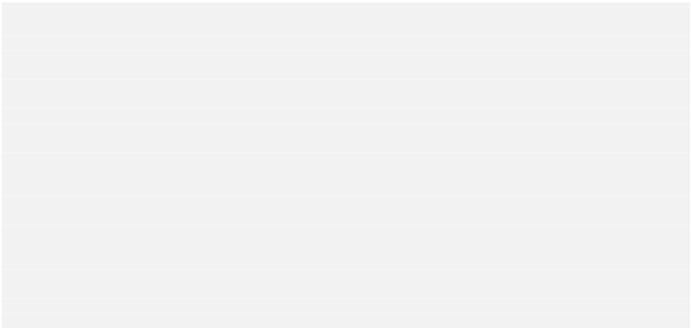
5

6

7

8

9



PostGIS 2.2.0dev 开发手册

570 / 686

10

11

12

(10 rows)

-- then add the face -

SELECT topology.AddFace(’ma\_topo’,

ST\_GeomFromText(’POLYGON((234896.5 899456.7,234914 899436.4,234946.6 899356.9,234872.5

899328.7,

234891 899285.4,234992.5 899145, 234890.6 899069,234755.2 899255.4,

234612.7 899379.4,234776.9 899563.7,234896.5 899456.7))’, 26986) ) As faceid;

-- result --

faceid

--------

1

**相关请参考**

AddEdge, CreateTopology, Section 4.3.1

←

**11.7.5**

**ST\_Simplify**

ST\_Simplify — 使用Douglas-Peucker 算法，将一个拓扑结构简化成简单的几何对象

**用法**

geometry ST\_Simplify(TopoGeometry geomA, ﬂoat tolerance);

**描述**

使用Douglas-Peucker 算法，将一个拓扑结构的每个边界简化成简单的几何对象

**注意**

返回的几何对象可能是非简单的或非有效的。分隔组成边界可能有助于保留简单性及有效性

该函数由GEOS模块提供支持.

可用版本：2.1.0

**相关请参考**

Geometry ST\_Simplify, ST\_IsSimple, ST\_IsValid, ST\_ModEdgeSplit

**11.8**

**11.8.1**

**拓扑结构构建函数**

**CreateTopoGeom**

CreateTopoGeom — 从一个拓扑元素数组创建一个拓扑结构。参数tg\_type值对应关系如下: 1:[multi]point, 2:[multi]line,

3:[multi]poly, 4:collection



PostGIS 2.2.0dev 开发手册

571 / 686

**用法**

topogeometry CreateTopoGeom(varchar toponame, integer tg\_type, integer layer\_id, topoelementarray tg\_objs);

topogeometry CreateTopoGeom(varchar toponame, integer tg\_type, integer layer\_id);

**描述**

为函数参数layer\_id标识的layer创建一个topogeometry对象，然后把它的信息注册到toposchema为toponame的关系表中。参数tg\_type是一个整型值，对应关系如下: 1:[multi]point, 2:[multi]line,3:[multi]poly, 4:collection

layer\_id是系统表topology.layer 的layer\_id字段。

点图层是由结点组成的，线性图层是由边集合组成的，面积图层是由面的集合组成的，集合图层是由结点，边，和面混合组成的。如果省略了第三个数组参数，那么将返回一个空的TopoGeometry 对象。

可用版本：1.?

**样例: Form from existing edges**

Create a topogeom in ri\_topo schema for layer 2 (our ri\_roads), of type (2) LINE, for the ﬁrst edge (we loaded in ST\_Create-

TopoGeo.

INSERT INTO ri.ri\_roads(road\_name, topo) VALUES(’Unknown’, topology.CreateTopoGeom(’ri\_topo ←

’,2,2,’{{1,2}}’::topology.topoelementarray);

**样例: Convert an areal geometry to best guess topogeometry**

Lets say we have geometries that should be formed from a collection of faces. We have for example blockgroups table and want

to know the topo geometry of each block group. If our data was perfectly aligned, we could do this:

-- create our topo geometry column --

SELECT topology.AddTopoGeometryColumn(

’topo\_boston’,

’boston’, ’blockgroups’, ’topo’, ’POLYGON’);

-- addtopgeometrycolumn --

1

-- update our column assuming

-- everything is perfectly aligned with our edges

UPDATE boston.blockgroups AS bg

SET topo = topology.CreateTopoGeom(’topo\_boston’

,3,1

, foo.bfaces)

FROM (SELECT b.gid,

topology.TopoElementArray\_Agg(ARRAY[f.face\_id,3]) As bfaces

FROM boston.blockgroups As b

INNER JOIN topo\_boston.face As f ON b.geom && f.mbr

WHERE ST\_Covers(b.geom, topology.ST\_GetFaceGeometry(’topo\_boston’, f.face\_id))

GROUP BY b.gid) As foo

WHERE foo.gid = bg.gid;

--the world is rarely perfect allow for some error

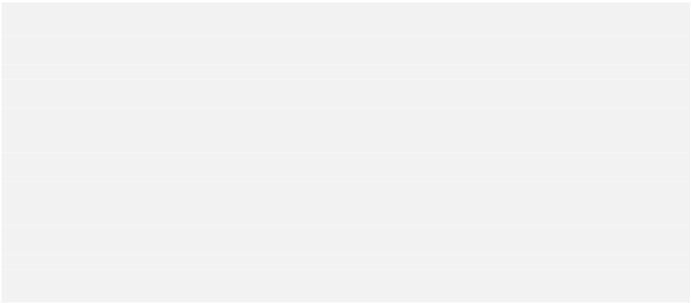
--count the face if 50% of it falls

-- within what we think is our blockgroup boundary

UPDATE boston.blockgroups AS bg

SET topo = topology.CreateTopoGeom(’topo\_boston’

,3,1



PostGIS 2.2.0dev 开发手册

572 / 686

, foo.bfaces)

FROM (SELECT b.gid,

topology.TopoElementArray\_Agg(ARRAY[f.face\_id,3]) As bfaces

FROM boston.blockgroups As b

INNER JOIN topo\_boston.face As f ON b.geom && f.mbr

WHERE ST\_Covers(b.geom, topology.ST\_GetFaceGeometry(’topo\_boston’, f.face\_id))

OR

(

ST\_Intersects(b.geom, topology.ST\_GetFaceGeometry(’topo\_boston’, f.face\_id))

AND ST\_Area(ST\_Intersection(b.geom, topology.ST\_GetFaceGeometry(’topo\_boston’,

←

f.face\_id) ) ) >

ST\_Area(topology.ST\_GetFaceGeometry(’topo\_boston’, f.face\_id))\*0.5

)

GROUP BY b.gid) As foo

WHERE foo.gid = bg.gid;

-- and if we wanted to convert our topogeometry back

-- to a denomalized geometry aligned with our faces and edges

-- cast the topo to a geometry

-- The really cool thing is my new geometries

-- are now aligned with my tiger street centerlines

UPDATE boston.blockgroups SET new\_geom = topo::geometry;

**相关请参考**

AddTopoGeometryColumn, toTopoGeom ST\_CreateTopoGeo, ST\_GetFaceGeometry, TopoElementArray, TopoElementArray\_Agg

**11.8.2**

**toTopoGeom**

toTopoGeom — 把一个简单的几何对象转换成拓扑结构

**用法**

topogeometry toTopoGeom(geometry geom, varchar toponame, integer layer\_id, ﬂoat8 tolerance);

topogeometry toTopoGeom(geometry geom, topogeometry topogeom, ﬂoat8 tolerance);

**描述**

把一个简单的几何对象转换成拓扑结构

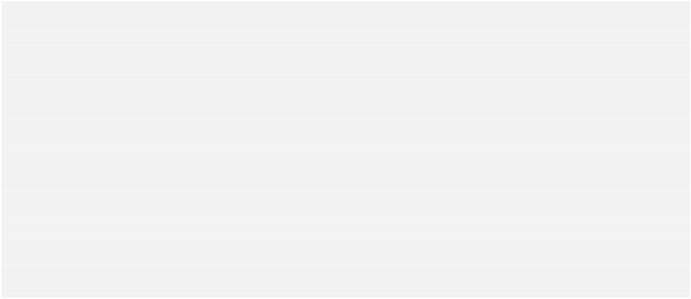
表示拓扑结构要素的输入geometry对象需要添加到底层的拓扑结构中，可能会分割掉已经存在的拓扑，并且他们需要在关系表中与输出的TopoGeometry产生关联。已经有的TopoGeometry对象(如果给出了topogeom参数值)会保留他们的形状。当参数tolerance给定了后，这个参数会用于把输入的geometry对象分割并装入到已经有拓扑要素中。在函数的第一种形式中，根据指定的拓扑结构toponame和图层layer\_id会创建一个新的TopoGeometry 对象。

在第二种形式中转换的结果会被添加到已经存在的TopoGeometry (topogeom)的对象中。

想要让新的形状完全替代就的形状参考clearTopoGeom.

可用版本：2.0

版本提升： 2.1.0 添加支持处理已经存在TopoGeometry对象功能



PostGIS 2.2.0dev 开发手册

573 / 686

**样例**

This is a full self-contained workﬂow

-- do this if you don’t have a topology setup already

-- creates topology not allowing any tolerance

SELECT topology.CreateTopology(’topo\_boston\_test’, 2249);

-- create a new table

CREATE TABLE nei\_topo(gid serial primary key, nei varchar(30));

--add a topogeometry column to it

SELECT topology.AddTopoGeometryColumn(’topo\_boston\_test’, ’public’, ’nei\_topo’, ’topo’, ’ ←

MULTIPOLYGON’) As new\_layer\_id;

new\_layer\_id

-----------

1

--use new layer id in populating the new topogeometry column

-- we add the topogeoms to the new layer with 0 tolerance

INSERT INTO nei\_topo(nei, topo)

SELECT nei,

topology.toTopoGeom(geom, ’topo\_boston\_test’, 1)

FROM neighborhoods

WHERE gid BETWEEN 1 and 15;

--use to verify what has happened --

SELECT \* FROM

topology.TopologySummary(’topo\_boston\_test’);

-- summary--

Topology topo\_boston\_test (5), SRID 2249, precision 0

61 nodes, 87 edges, 35 faces, 15 topogeoms in 1 layers

Layer 1, type Polygonal (3), 15 topogeoms

Deploy: public.nei\_topo.topo

-- Shrink all TopoGeometry polygons by 10 meters

UPDATE nei\_topo SET topo = ST\_Buffer(clearTopoGeom(topo), -10);

-- Get the no-one-lands left by the above operation

-- I think GRASS calls this "polygon0 layer"

SELECT ST\_GetFaceGeometry(’topo\_boston\_test’, f.face\_id)

FROM topo\_boston\_test.face f

WHERE f.face\_id > 0 -- don’t consider the universe face

AND NOT EXISTS ( -- check that no TopoGeometry references the face

SELECT \* FROM topo\_boston\_test.relation

WHERE layer\_id = 1 AND element\_id = f.face\_id

);

**相关请参考**

CreateTopology, AddTopoGeometryColumn, CreateTopoGeom, TopologySummary, clearTopoGeom

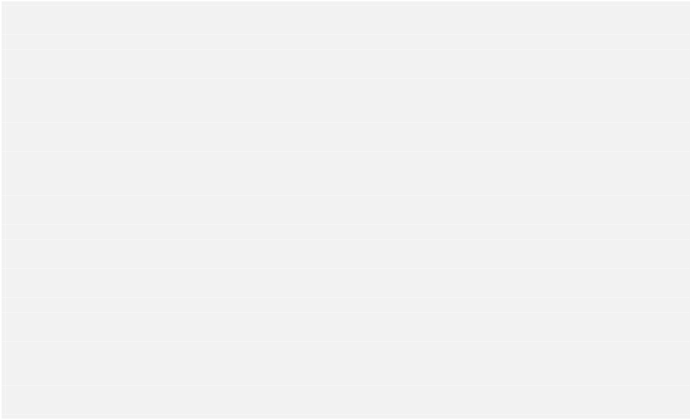
**11.8.3**

**TopoElementArray\_Agg**

TopoElementArray\_Agg —返回一个topoelementarray数组，它是element\_id，type arrays (topoelements)的集合。

**用法**

topoelementarray TopoElementArray\_Agg(topoelement set teﬁeld);



PostGIS 2.2.0dev 开发手册

574 / 686

**描述**

这个函数用于从TopoElement集合中创建一个TopoElementArray 对象

可用版本：2.0.0

**样例**

SELECT topology.TopoElementArray\_Agg(ARRAY[e,t]) As tea

FROM generate\_series(1,3) As e CROSS JOIN generate\_series(1,4) As t;

tea

--------------------------------------------------------------------------

{{1,1},{1,2},{1,3},{1,4},{2,1},{2,2},{2,3},{2,4},{3,1},{3,2},{3,3},{3,4}}

**相关请参考**

TopoElement, TopoElementArray

**11.9**

**11.9.1**

**拓扑结构编辑函数**

**clearTopoGeom**

clearTopoGeom —清除拓扑结构的内容

**用法**

topogeometry clearTopoGeom(topogeometry topogeom);

**描述**

清除拓扑结构的内容。通常和函数TopoGeom一起使用，用于替换已经存在的对象的外形和任何更高层次依赖的对象。

可用版本：2.1

**样例**

-- Shrink all TopoGeometry polygons by 10 meters

UPDATE nei\_topo SET topo = ST\_Buffer(clearTopoGeom(topo), -10);

**相关请参考**

toTopoGeom

**11.9.2**

**toTopoGeom**

toTopoGeom — 添加一个geometry 形状到给定的拓扑结构中

**描述**

参考 toTopoGeom

PostGIS 2.2.0dev 开发手册

575 / 686

**11.10**

**11.10.1**

**拓扑结构存取函数**

**GetTopoGeomElementArray**

GetTopoGeomElementArray —返回一个topoelementarray对象 (一个元素是topoelements的数组)包含了拓扑元素和给定TopoGeometry对象的类型 (要素元素)

**用法**

topoelementarray GetTopoGeomElementArray(varchar toponame, integer layer\_id, integer tg\_id);

topoelementarray topoelement GetTopoGeomElementArray(topogeometry tg);

**描述**

返回一个TopoElementArray对象 (一个元素是topoelements的数组)包含了拓扑元素和给定TopoGeometry对象的类型 (要素元素)。这个函数和函数GetTopoGeomElements 很类似，区别在于这个函数返回的是元素的数组而不是一个数据集。

tg\_id是系统表topology.layer的layer\_id字段标识的layer图层中的拓扑结构的ID

可用版本：1.?

**样例**

**相关请参考**

GetTopoGeomElements, TopoElementArray

**11.10.2**

**GetTopoGeomElements**

GetTopoGeomElements —返回一个给定的拓扑结构的拓扑元素对象，返回值包含两部分：element\_id和element\_type(要素类型)

**用法**

setof topoelement GetTopoGeomElements(varchar toponame, integer layer\_id, integer tg\_id);

setof topoelement GetTopoGeomElements(topogeometry tg);

**描述**

返回一个给定的拓扑结构的拓扑元素对象，返回值包含两部分：element\_id和element\_type(要素类型)

tg\_id是系统表topology.layer的layer\_id字段标识的layer图层中的拓扑结构的ID

可用版本：1.?

**样例**

**相关请参考**

GetTopoGeomElementArray, TopoElement

PostGIS 2.2.0dev 开发手册

576 / 686

**11.11**

**11.11.1**

**拓扑结构的输出**

**AsGML**

AsGML — 返回一个拓扑结构的GML格式输出

**用法**

text AsGML(topogeometry tg);

text AsGML(topogeometry tg, text nspreﬁx\_in);

text AsGML(topogeometry tg, regclass visitedTable);

text AsGML(topogeometry tg, regclass visitedTable, text nspreﬁx);

text AsGML(topogeometry tg, text nspreﬁx\_in, integer precision, integer options);

text AsGML(topogeometry tg, text nspreﬁx\_in, integer precision, integer options, regclass visitedTable);

text AsGML(topogeometry tg, text nspreﬁx\_in, integer precision, integer options, regclass visitedTable, text idpreﬁx);

text AsGML(topogeometry tg, text nspreﬁx\_in, integer precision, integer options, regclass visitedTable, text idpreﬁx, int gm-

lversion);

**描述**

返回一个拓扑结构的GML3格式输出。如果没有指定nsprefix\_in参数，该值就设置为gml，如果设置该值为空，则得到没有前缀的namespace prefix。

参数 precision (默认值: 15) 和参数 options (默认值 1)parameters如果提供了的话，会原值传递给被调用的函数ST\_AsGML.

参数 visitedTable 如果提供了的话，用于跟踪访问节点和边缘元素，以便使用交叉引用(xlink:xref)方式而不是复制定义的方式。

参数代表的表至少要有两个整型字段： ’element\_type’和 ’element\_id’。调用该函数的用户必须在该表上有读写权限。为了最佳的性能考虑看，应该在字段element\_type 和 element\_id建立索引。这样的索引可用通过在这两个字段上面建立唯一约束的方式来自动创建索引。（译者注：在PostgreSQL中primary key相当于索引）

样例:

CREATE TABLE visited (

element\_type integer, element\_id integer,

unique(element\_type, element\_id)

);

参数 idprefix 参数如果提供了的话，会被当做边界和点的标签标识符

参数 gmlver 如果提供了的话，会被传给函数ST\_AsGML，默认值为3

可用版本：2.0.0

**样例**

This uses the topo geometry we created in CreateTopoGeom

SELECT topology.AsGML(topo) As rdgml

FROM ri.roads

WHERE road\_name = ’Unknown’;

-- rdgml--

<gml:TopoCurve>

<gml:directedEdge>

<gml:Edge gml:id="E1">

<gml:directedNode orientation="-">

<gml:Node gml:id="N1"/>

</gml:directedNode>

<gml:directedNode></gml:directedNode>

<gml:curveProperty>

PostGIS 2.2.0dev 开发手册

577 / 686

<gml:Curve srsName="urn:ogc:def:crs:EPSG::3438">

<gml:segments>

<gml:LineStringSegment>

<gml:posList srsDimension="2">384744 236928 384750 236923

384769 236911 384799 236895 384811 236890

←

384833 236884 384844 236882 384866 236881 384879 236883 384954

236898 385087 236932 385117 236938

385167 236938 385203 236941 385224 236946 385233 236950 385241

236956 385254 236971

385260 236979 385268 236999 385273 237018 385273 237037 385271

237047 385267 237057 385225 237125

385210 237144 385192 237161 385167 237192 385162 237202 385159

237214 385159 237227 385162 237241

385166 237256 385196 237324 385209 237345 385234 237375 385237

237383 385238 237399 385236 237407

385227 237419 385213 237430 385193 237439 385174 237451 385170

237455 385169 237460 385171 237475

385181 237503 385190 237521 385200 237533 385206 237538 385213

237541 385221 237542 385235 237540 385242 237541

385249 237544 385260 237555 385270 237570 385289 237584 385292

237589 385291 237596 385284 237630</gml:posList>

</gml:LineStringSegment>

</gml:segments>

</gml:Curve>

</gml:curveProperty>

</gml:Edge>

</gml:directedEdge>

</gml:TopoCurve>

Same exercise as previous without namespace

SELECT topology.AsGML(topo,’’) As rdgml

FROM ri.roads

WHERE road\_name = ’Unknown’;

-- rdgml--

<TopoCurve>

<directedEdge>

<Edge id="E1">

<directedNode orientation="-">

<Node id="N1"/>

</directedNode>

<directedNode></directedNode>

<curveProperty>

<Curve srsName="urn:ogc:def:crs:EPSG::3438">

<segments>

<LineStringSegment>

<posList srsDimension="2">384744 236928 384750 236923 384769

236911 384799 236895 384811 236890

384833 236884 384844 236882 384866 236881 384879 236883 384954

236898 385087 236932 385117 236938

385167 236938 385203 236941 385224 236946 385233 236950 385241

236956 385254 236971

385260 236979 385268 236999 385273 237018 385273 237037 385271

237047 385267 237057 385225 237125

385210 237144 385192 237161 385167 237192 385162 237202 385159

237214 385159 237227 385162 237241

385166 237256 385196 237324 385209 237345 385234 237375 385237

237383 385238 237399 385236 237407

385227 237419 385213 237430 385193 237439 385174 237451 385170

237455 385169 237460 385171 237475

←

←

←

←

←

←

←

←

←

←

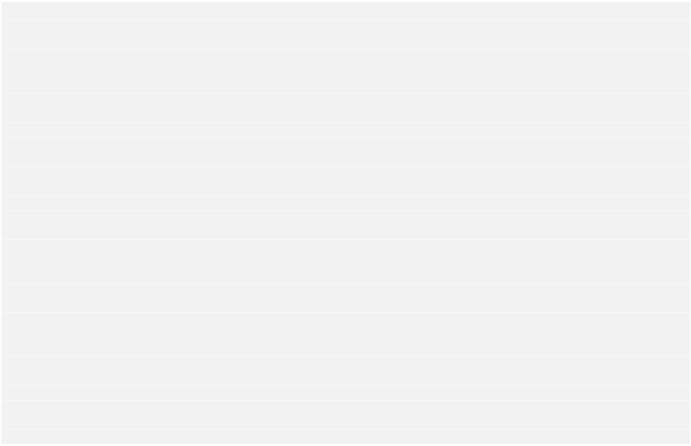
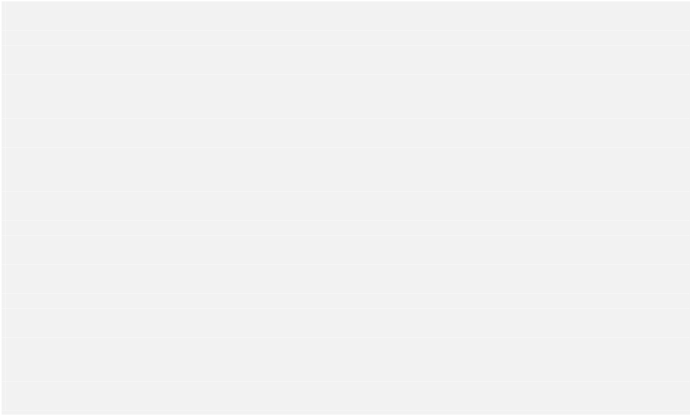
←

←

←

←

←



PostGIS 2.2.0dev 开发手册

578 / 686

385181 237503 385190 237521 385200 237533 385206 237538 385213

237541 385221 237542 385235 237540 385242 237541

385249 237544 385260 237555 385270 237570 385289 237584 385292

237589 385291 237596 385284 237630</posList>

</LineStringSegment>

</segments>

</Curve>

</curveProperty>

</Edge>

</directedEdge>

</TopoCurve>

**相关请参考**

CreateTopoGeom, ST\_CreateTopoGeo

←

←

**11.11.2**

**AsTopoJSON**

AsTopoJSON — 返回描述一个拓扑结构的TopoJSON格式描述

**用法**

text AsTopoJSON(topogeometry tg, regclass edgeMapTable);

**描述**

返回描述一个拓扑结构的TopoJSON格式描。参数edgeMapTable不是空值 ，它会被用来查找或存储边ID与arc index的映射关系。

这是为了能够允许在最终生成的topojson格式的文档中的形成一个紧凑的“弧”数组。如果提供了table参数，那么需要有一个名为“arc\_id”类型为“serial” 的字段和一个名为“edge\_id”的整型字段；函数将会查询表的“edge\_id”字段，因此建议在该字段上加上索引。

**注意**

TopoJSON 格式输出的arc 部分下标从0开始，但是在表"edgeMapTable" 中下标从1开始。

一个完整的TopoJSON 文档需要包含除了该函数返回的片段外，实际的arcs部分还加上了一下header头信息，请参考 [TopoJSON speciﬁcation.](http://github.com/mbostock/topojson/wiki/Specification)

可用版本：2.1.0

**相关请参考**

ST\_AsGeoJSON

**样例**

CREATE TEMP TABLE edgemap(arc\_id serial, edge\_id int unique);

-- header

SELECT ’{ "type": "Topology", "transform": { "scale": [1,1], "translate": [0,0] }, "objects ←

": {’



PostGIS 2.2.0dev 开发手册

579 / 686

-- objects

UNION ALL SELECT ’"’ || feature\_name || ’": ’ || AsTopoJSON(feature, ’edgemap’)

FROM features.big\_parcels WHERE feature\_name = ’P3P4’;

-- arcs

WITH edges AS (

SELECT m.arc\_id, e.geom FROM edgemap m, city\_data.edge e

WHERE e.edge\_id = m.edge\_id

), points AS (

SELECT arc\_id, (st\_dumppoints(geom)).\* FROM edges

), compare AS (

SELECT p2.arc\_id,

CASE WHEN p1.path IS NULL THEN p2.geom

ELSE ST\_Translate(p2.geom, -ST\_X(p1.geom), -ST\_Y(p1.geom))

END AS geom

FROM points p2 LEFT OUTER JOIN points p1

ON ( p1.arc\_id = p2.arc\_id AND p2.path[1] = p1.path[1]+1 )

ORDER BY arc\_id, p2.path

), arcsdump AS (

SELECT arc\_id, (regexp\_matches( ST\_AsGeoJSON(geom), ’\[.\*\]’))[1] as t

FROM compare

), arcs AS (

SELECT arc\_id, ’[’ || array\_to\_string(array\_agg(t), ’,’) || ’]’ as a FROM arcsdump

GROUP BY arc\_id

ORDER BY arc\_id

)

SELECT ’}, "arcs": [’ UNION ALL

SELECT array\_to\_string(array\_agg(a), E’,\n’) from arcs

-- footer

UNION ALL SELECT ’]}’::text as t;

-- Result:

{ "type": "Topology", "transform": { "scale": [1,1], "translate": [0,0] }, "objects": {

"P3P4": { "type": "MultiPolygon", "arcs": [[[-1]],[[6,5,-5,-4,-3,1]]]}

}, "arcs": [

[[25,30],[6,0],[0,10],[-14,0],[0,-10],[8,0]],

[[35,6],[0,8]],

[[35,6],[12,0]],

[[47,6],[0,8]],

[[47,14],[0,8]],

[[35,22],[12,0]],

[[35,14],[0,8]]

]}

**11.12**

**11.12.1**

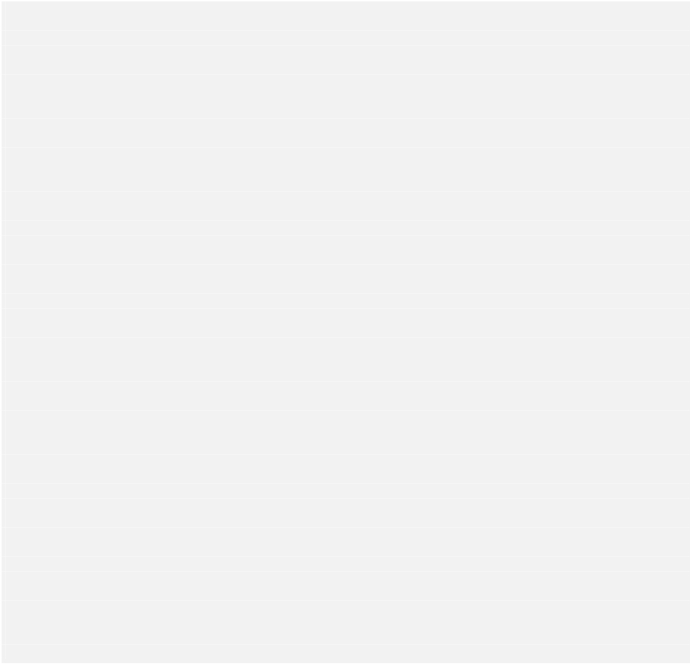
**拓扑空间关系函数**

**Equals**

Equals — 如果两个拓扑结构是由基本的拓扑要素组成，则返回true

**用法**

boolean Equals(topogeometry tg1, topogeometry tg2);



PostGIS 2.2.0dev 开发手册

580 / 686

**描述**

如果两个拓扑结构是由基本的拓扑要素，比如面、边、点组成，则返回true

**注意**

该函数不支持拓扑结构是geometry collection的类型。该函数也不能比较不同拓扑结构的拓扑结构。目前也不支持层级的拓扑结构（层级拓扑结构：是指拓扑结构是由其他拓扑结构组成的）.

可用版本：1.x

该函数支持3D对象，并且不会丢弃Z坐标.

**样例**

**相关请参考**

GetTopoGeomElements, ST\_Equals

**11.12.2**

**Intersects**

Intersects — 如果两个基本的拓扑结构是由基本的拓扑要素组成，那么返回true

**用法**

boolean Equals(topogeometry tg1, topogeometry tg2);

**描述**

如果两个基本的拓扑结构是由基本的拓扑要素组成，那么返回true

**注意**

该函数不支持拓扑结构是geometry collection的类型。该函数也不能比较不同拓扑结构的拓扑结构。目前也不支持层级的拓扑结构（层级拓扑结构：是指拓扑结构是由其他拓扑结构组成的）

可用版本：1.?

该函数支持3D对象，并且不会丢弃Z坐标.

**样例**

**相关请参考**

ST\_Intersects

