# Chapter 8

**PostGIS 参考**

下面的提供的函数是那些PostGIS用户可能需要的。还有其他的函数可能一般用户没法用到

**注意**

PostGIS已经开始从现有的命名惯例转变为SQL-MM-centric规范。因此，大部分你知道的函数，已经使用标准的空间类型（ST\_）前缀来重命名.先前的函数仍然是可用的，虽然已经被等价的更新命名方式的函数替代，但并没有列出他们。没有在本文档列出的非ST\_函数格式的函数是过时的，并且在未来版本将被删除。因此请停止使用这些函数.

**8.1**

**8.1.1**

**PostgreSQL PostGIS Geometry(几何)/Geography(地理)/Box(矩形)类型**

**box2d**

box2d —box2d是一中包含 x min, ymin, xmax, ymax的二维几何类型，反映封闭箱子状几何对象的范围.

**描述**

box2d是一种用于表示包裹一个几何对象或一系列几何对象的封闭“箱子”空间数据类型。

ST\_Extent函数在PostGIS 1.4版本之前的更早版本会返回一个box2d对象

**8.1.2**

**box3d**

box3d —box3d是postgis包含 x min, ymin, zmin, xmax, ymax， zmax，反映三维的几何类型，通常用来返回一个几何对象或一系列几何对象的3维扩展区域。

**描述**

box3d是一种用于表示包裹一个几何对象或一系列几何对象的封闭“箱子”空间数据类型。 ST\_3DExtent 返回一个box3d 对象

**转换方式**

这一节列出了这种类型自动也可以是显式地转换成其他类型的方式：



80 / 686

**8.1.3**

译者注：例如：**SELECT** box3d(Box2D(ST\_GeomFromText('LINESTRING(1 2, 3 4, 5 6)')));

**geometry**

geometry — 二维空间数据类型.

**描述**

geometry 是一个基本的 postgis空间数据类型，用于表示欧几里得空间坐标系的一种几何特征(译者注：例如点、线等等)

**转换方式**

这一节列出了这种类型自动也可以是显式地转换成其他类型的方式：

**相关请参考**

Section 4.1即4.1节

**8.1.4**

**geometry\_dump**

一种包含geom和路径两个字段的空间数据类型。-geom(包含一个几何类型对象)，path[]（是一个含有被导出的几何对象位置信息的1-d(d表示维度)数组）是一个复合数据类型，可以根据数组原理，导航获取路径某个位置的元素，可以反映一个复杂的几何对象的组合部分和定位部分。

**描述**

一种包含geom和path[]路径数组两个字段的复合数据类型。path[]是一个1-维度值(数组下标从1开始计数，例如path[1]获取第一个元素)的数组。该数组定义了转储的几何对象的导航路径。ST\_Dump\*系列的函数使用该类型的几何对象作为输出类型，以便把一个更复杂的几何对象分解为的组合部分和定位部分。

**相关请参考**

Section 13.5

**8.1.5**

**geography**

geography — 椭圆空间数据类型.

|  |  |
| --- | --- |
| 转换成的目标类型 | 转换方式 |
| box | 自动 |
| box2d | 自动 |
| box3d | 自动 |
| bytea | 自动 |
| geography | 自动 |
| text | 自动 |

|  |  |
| --- | --- |
| 目标类型 | 转换方式 |
| box | 自动 |
| box2d | 自动 |
| geometry | 自动 |

81 / 686

**描述**

geography是一种在地球参考系中代表一种几何对象的空间数据类型

**转换方式**

这一节列出了这种类型自动也可以是显式地转换成其他类型的方式：（翻译待确定）

**相关请参考**

Section 13.3, Section 4.2

**8.2**

**8.2.1**

**PostGIS 统一自定义变量 (GUCs)**

**postgis.backend变量**

postgis.backend — 该变量用于服务端程序选择geos或sfcgal库. 变量值选项: geos 或 sfcgal. 默认是geos.

**描述**

GUC只在你编译PostGIS时添加了sfcgal支持有关。默认情况下在GEOS和SFCGAL有同名函数时候，使用geos提供的函数做支持。这个变量允许你选择SFCGAL做后端服务请求支持（即不选择GEOS做支持

可用PostGIS版本: 2.1.0

**相关请参考**

Section 8.10

**8.2.2**

**postgis.gdal\_datapath**

postgis.gdal\_datapath—用来指定GDAL的GDAL\_DATA值。如果没有设置，运行环境将设置GDAL\_DATA这个变量已被使用

**描述**

一种设置GDAL的GDAL\_DATA选项值的GUC变量。postgis.gdal\_datapath应该是GDAL数据文件的完整的物理路径。

这个选项最长用在GDAL的数据文件路径不是硬编码的Windows平台上。这个选项在GDAL数据文件没有在GDAL默认路径

时候，也需要显式地设置。

**注意**

该参数可以在PostgreSQL的配置文件postgresql.conf中设置，这个参数也可以在连接数据库时候或者事务中设置



|  |  |
| --- | --- |
| 目标类型 | 转换方式 |
| geometry | 显式转换 |

82 / 686

可用版本: 2.2.0

**注意**

关于更多的GDAL\_DATA信息，请参考[Conﬁguration Options](http://trac.osgeo.org/gdal/wiki/ConfigOptions)(http://trac.osgeo.org/gdal/wiki/ConfigOptions)[.](http://trac.osgeo.org/gdal/wiki/ConfigOptions)

**样例**

设置 postgis.gdal\_datapath或将该参数设置为默认值方法：

SET postgis.gdal\_datapath TO ‘/usr/local/share/gdal.hidden’;

SET postgis.gdal\_datapath TO default;

在Windows平台上对指定的数据库设置该值

ALTER DATABASE gisdb

SET postgis.gdal\_datapath = ‘C:/Program Files/PostgreSQL/9.3/gdal-data’;

**相关请参考**

PostGIS\_GDAL\_Version, ST\_Transform

**8.3**

**8.3.1**

**管理函数**

**AddGeometryColumn**

AddGeometryColumn —添加一个几何类型的列到一个已经存在的表中。默认使用类型修改器来定义字段而不是约束器。可以传递一个false逻辑值给use\_typmod参数来获取基于老的约束条件方式。

**用法**

text AddGeometryColumn(varchar table\_name, varchar column\_name, integer srid, varchar type, integer dimension, boolean

use\_typmod=true);

text AddGeometryColumn(varchar schema\_name, varchar table\_name, varchar column\_name, integer srid, varchar type, inte-

ger dimension, boolean use\_typmod=true);

text AddGeometryColumn(varchar catalog\_name, varchar schema\_name, varchar table\_name, varchar column\_name, integer

srid, varchar type, integer dimension, boolean use\_typmod=true);

**描述**

添加一个几何类型的列到一个已知的表中。schema\_name就是表的schema\_name. srid必须是PostGIS的系统表SPATIAL\_REF\_SYS中存在的整型值。如果schema\_name不存在或者在当前search\_path值不可见(译者注：search\_path值可以设置，如果添加该列时候，所需要的schema\_name，恰好在当前使用的schema\_name里面，那么这个值可以不设置，否则就是需要设置的)，具体的SRID、geometry类型或维度值无效，都会抛出异常。

**注意**

2.0.0版本变更:这个函数不再更新geometry\_columns中的记录，因为geometry\_columns现在是从system catalogs(pg\_catalog)读取的数据的视图。该函数默认也不创建约束，相反它使用PostgreSQL内置的类型语句修改方式。因此例如创建一个wgs94 POINT类型的列，等价于这个语句: ALTER TABLEsome\_table ADD COLUMN geom geometry(Point,4326);

2.0.0版本更新：如果你想使用旧的约束方式，使用use\_typmod参数，但默认该值设置为false





83 / 686

**注意**

2.0.0版本更新：视图不能再手动在geometry\_columns中更新信息，然而依赖geometry typmod 方式建的表和没有使用wrapper 函数的视图将会自动注册信息，因为他们继承了他们父表的typmod方式。那些视图使用几何类型函数输出其他的几何类型对象需要被转换成typmod的几何类型对象，以便这些视图中的几何类型的能够在geometry\_columns这个视图中正确注册(就是查询时候能得到这个视图包含的几何类型的记录)

参考 Section 4.3.4.

该方法实现了OpenGIS 关于SQL 1.1的简单特性实现规范 [OpenGISSimple Features Implementation Speciﬁcation for SQL 1.1.](http://www.opengeospatial.org/standards/sfs)

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

该方法支持CircularString和curve几何类型字段

提升: 2.0.0版本引进了use\_typmod参数，默认使用typmod方式（即用创建一般表字段的方式创建）创建几何类型字段而不是基于约束(基于约束的方法就是用AddGeometryColumn函数的方式)的方式创建。

**样例**

-- 创建一个schema

CREATE SCHEMA my\_schema;

-- 创建一个PostgreSQL表

CREATE TABLE my\_schema.my\_spatial\_table (id serial);

-- Describing the table shows a simple table with a single "id" column.

postgis=# \d my\_schema.my\_spatial\_table

Table "my\_schema.my\_spatial\_table"

Column |

Type

|

Modifiers

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

id

| integer | not null default nextval(‘my\_schema.my\_spatial\_table\_id\_seq’::regclass)

-- Add a spatial column to the table

SELECT AddGeometryColumn (‘my\_schema’,’my\_spatial\_table’,’geom’,4326,’POINT’,2);

-- Add a point using the old constraint based behavior

SELECT AddGeometryColumn (‘my\_schema’,’my\_spatial\_table’,’geom\_c’,4326,’POINT’,2, false);

--Add a curvepolygon using old constraint behavior

SELECT AddGeometryColumn (‘my\_schema’,’my\_spatial\_table’,’geomcp\_c’,4326,’CURVEPOLYGON’,2,

false);

-- Describe the table again reveals the addition of a new geometry columns.

\d my\_schema.my\_spatial\_table

addgeometrycolumn

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

my\_schema.my\_spatial\_table.geomcp\_c SRID:4326 TYPE:CURVEPOLYGON DIMS:2

(1 row)

Table "my\_schema.my\_spatial\_table"

Column

|

Type

|

Modifiers

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

id

| integer

| not null default nextval(‘my\_schema.

my\_spatial\_table\_id\_seq’::regclass)

geom

| geometry(Point,4326) |

geom\_c

| geometry

|

geomcp\_c | geometry

Check constraints:

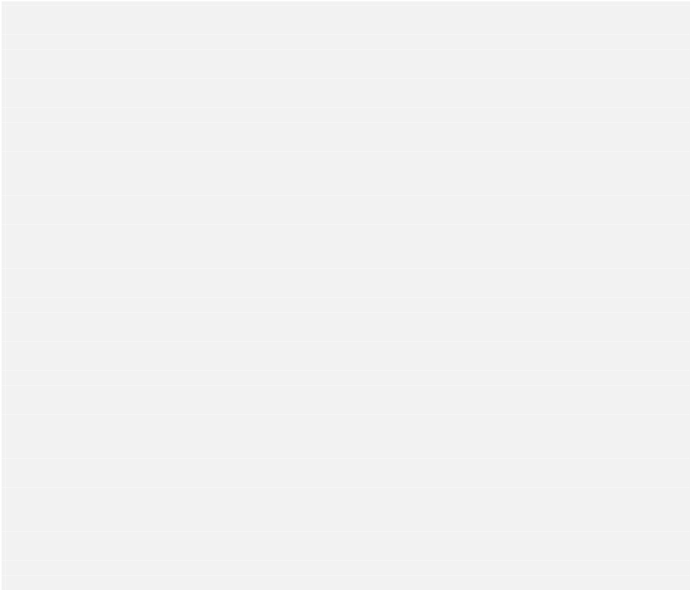
|











84 / 686

"enforce\_dims\_geom\_c" CHECK (st\_ndims(geom\_c) = 2)

"enforce\_dims\_geomcp\_c" CHECK (st\_ndims(geomcp\_c) = 2)

"enforce\_geotype\_geom\_c" CHECK (geometrytype(geom\_c) = ‘POINT’::text OR geom\_c IS NULL)

"enforce\_geotype\_geomcp\_c" CHECK (geometrytype(geomcp\_c) = ‘CURVEPOLYGON’::text OR

geomcp\_c IS NULL)

"enforce\_srid\_geom\_c" CHECK (st\_srid(geom\_c) = 4326)

"enforce\_srid\_geomcp\_c" CHECK (st\_srid(geomcp\_c) = 4326)

-- geometry\_columns view also registers the new columns --

SELECT f\_geometry\_column As col\_name, type, srid, coord\_dimension As ndims

FROM geometry\_columns

WHERE f\_table\_name = ‘my\_spatial\_table’ AND f\_table\_schema = ‘my\_schema’;

col\_name |

type

| srid | ndims

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

geom

geom\_c

| Point

| Point

| 4326 | 2

| 4326 | 2

geomcp\_c | CurvePolygon | 4326 | 2

**相关请参考**

DropGeometryColumn, DropGeometryTable, Section 4.3.2, Section 4.3.4

**8.3.2**

**DropGeometryColumn**

DropGeometryColumn — 从一个空间表中删除一个几何类型的列

**用法：**

text DropGeometryColumn(varchar table\_name, varchar column\_name);

text DropGeometryColumn(varchar schema\_name, varchar table\_name, varchar column\_name);

text DropGeometryColumn(varchar catalog\_name, varchar schema\_name, varchar table\_name, varchar column\_name);

**描述**

从一个空间表中删除一个几何类型的列.注意schema\_name是需要的，为了匹配geometry\_columns表的几何类型列。

译者备注：一个数据库可以有多个schema，所以为了精确表达一个字段，正确的方式是schema\_name.table\_name.column\_name

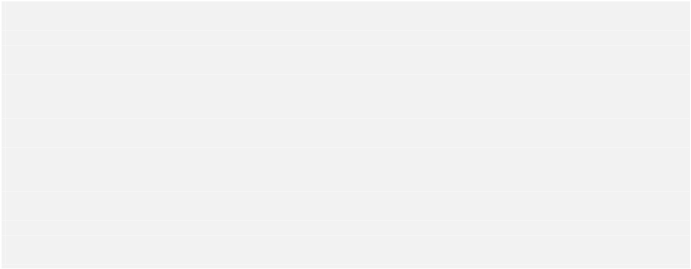
该方法实现了OpenGIS 关于SQL 1.1的简单特性实现规范 [OpenGIS Simple Features Implementation Speciﬁcation for SQL 1.1.](http://www.opengeospatial.org/standards/sfs)

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

该方法支持CircularString和curve几何类型字段

**注意**

变更: 2.0.0 版本该函数是提供了后向兼容。既然geometry\_columns是一个依赖于system catalog(译者注：即pg\_catalog)数据信息的视图，因此你可以使用ALTER TABLE这种语法像其他一般类型表的列一样删除一个带有几何类型的列。











85 / 686

**样例**

SELECT DropGeometryColumn (‘my\_schema’,’my\_spatial\_table’,’geom’);

----RESULT output ---

dropgeometrycolumn

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

my\_schema.my\_spatial\_table.geom effectively removed.

-- 在 PostGIS 2.0+ 上面等价于下面的方法

-- the standard alter table.

Both will deregister from geometry\_columns

ALTER TABLE my\_schema.my\_spatial\_table DROP column geom;

**相关请参考**

AddGeometryColumn, DropGeometryTable, Section 4.3.2

**8.3.3**

**DropGeometryTable**

DropGeometryTable — 删除一个表及其在geometry\_columns 中的引用(译者注：每一个含有几何类型字段的表都会在geometry\_columns中存有记录).

**语法**

boolean DropGeometryTable(varchar table\_name);

boolean DropGeometryTable(varchar schema\_name, varchar table\_name);

boolean DropGeometryTable(varchar catalog\_name, varchar schema\_name, varchar table\_name);

**描述**

删除一个表及其在geometry\_columns 中的引用. 注意：如果该函数没有设置schema\_name的值，将使用: current\_schema() 的值作为schema\_name(译者注：即select current\_schema()得到的值)

**注意**

变更: 2.0.0 版本该函数是提供了后向兼容。既然geometry\_columns是一个依赖于system catalog(译者注：即pg\_catalog)数据信息的视图，因此你可以使用DROP TABLE这种语法像其他一般类型的表一样删除一个带有几何类型的表。

**样例**

SELECT DropGeometryTable (‘my\_schema’,’my\_spatial\_table’);

----RESULT output ---

my\_schema.my\_spatial\_table dropped.

-- The above is now equivalent to --

DROP TABLE my\_schema.my\_spatial\_table;

**相关请参考**

AddGeometryColumn, DropGeometryColumn, Section 4.3.2

**8.3.4**

**PostGIS\_完整版本**

PostGIS\_Full\_Version — 获取postgis的完整版本信息和编译配置信息.



86 / 686

**语法**

text PostGIS\_Full\_Version();

**描述**

获取postgis的完整版本信息和编译配置信息.同时通知关于库和脚本的一致性，建议根据需要升级

**样例**

SELECT PostGIS\_Full\_Version();

postgis\_full\_version

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

POSTGIS="1.3.3" GEOS="3.1.0-CAPI-1.5.0" PROJ="Rel. 4.4.9, 29 Oct 2004" USE\_STATS

(1 row)

**相关请参考**

Section 2.9, PostGIS\_GEOS\_Version, PostGIS\_Lib\_Version, PostGIS\_LibXML\_Version, PostGIS\_PROJ\_Version, PostGIS\_Version

**8.3.5**

**PostGIS\_GEOS\_Version**

PostGIS\_GEOS\_Version — 返回GEOS 库的版本号.

**语法**

text PostGIS\_GEOS\_Version();

**描述**

返回GEOS 库的版本号, 如果没有GEOS开启支持(没有安装)，返回NULL.

**样例**

SELECT PostGIS\_GEOS\_Version();

postgis\_geos\_version

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

3.1.0-CAPI-1.5.0

(1 row)

**相关请参考**

PostGIS\_Full\_Version, PostGIS\_Lib\_Version, PostGIS\_LibXML\_Version, PostGIS\_PROJ\_Version, PostGIS\_Version

**8.3.6**

**PostGIS\_LibXML\_Version**

PostGIS\_LibXML\_Version — 返回 libxml2 库的版本号.

87 / 686

**语法**

text PostGIS\_LibXML\_Version();

**描述**

返回 libxml2 库的版本号.

可用postgis版本: 1.5

**样例**

SELECT PostGIS\_LibXML\_Version();

postgis\_libxml\_version

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

2.7.6

(1 row)

**相关请参考**

PostGIS\_Full\_Version, PostGIS\_Lib\_Version, PostGIS\_PROJ\_Version, PostGIS\_GEOS\_Version, PostGIS\_Version

**8.3.7**

**PostGIS\_Lib\_Build\_Date**

PostGIS\_Lib\_Build\_Date — 返回 PostGIS 库的构建日期.

**语法**

text PostGIS\_Lib\_Build\_Date();

**描述**

返回 PostGIS 库的构建日期.

**样例**

SELECT PostGIS\_Lib\_Build\_Date();

postgis\_lib\_build\_date

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

2008-06-21 17:53:21

(1 row)

**8.3.8**

**PostGIS\_Lib\_Version**

PostGIS\_Lib\_Version — 返回PostGIS库的版本号.

**语法**

text PostGIS\_Lib\_Version();

88 / 686

**描述**

返回PostGIS库的版本号.

**样例**

SELECT PostGIS\_Lib\_Version();

postgis\_lib\_version

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

1.3.3

(1 row)

**相关请参考**

PostGIS\_Full\_Version, PostGIS\_GEOS\_Version, PostGIS\_LibXML\_Version, PostGIS\_PROJ\_Version, PostGIS\_Version

**8.3.9**

**PostGIS\_PROJ\_Version**

PostGIS\_PROJ\_Version —返回PROJ4 库的版本号.

**用法**

text PostGIS\_PROJ\_Version();

**描述**

返回PROJ4 库的版本号，如果返回NULL表示没有开启PROJ4 支持

**样例**

SELECT PostGIS\_PROJ\_Version();

postgis\_proj\_version

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

Rel. 4.4.9, 29 Oct 2004

(1 row)

**相关请参考**

PostGIS\_Full\_Version, PostGIS\_GEOS\_Version, PostGIS\_Lib\_Version, PostGIS\_LibXML\_Version, PostGIS\_Version

**8.3.10**

**PostGIS\_Scripts\_Build\_Date**

PostGIS\_Scripts\_Build\_Date — 返回PostGIS脚本的编译日期.

**语法**

text PostGIS\_Scripts\_Build\_Date();

89 / 686

**描述**

返回PostGIS脚本的编译日期.

Availability: 1.0.0RC1

**样例**

SELECT PostGIS\_Scripts\_Build\_Date();

postgis\_scripts\_build\_date

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

2007-08-18 09:09:26

(1 row)

**相关请参考**

PostGIS\_Full\_Version, PostGIS\_GEOS\_Version, PostGIS\_Lib\_Version, PostGIS\_LibXML\_Version, PostGIS\_Version

**8.3.11**

**PostGIS\_脚本安装函数**

PostGIS\_Scripts\_Installed — 返回安装于数据库的postgis脚本版本.

**语法**

text PostGIS\_Scripts\_Installed();

**描述**

返回安装于数据库的postgis脚本版本.

**注意**

如果这个函数的输出和PostGIS\_Scripts\_Released的输出不一致，你看可能没有正确地升级。请查看升级章节获取更多信息。

可用版本:至少postgis 0.9.0+

**样例**

SELECT PostGIS\_Scripts\_Installed();

postgis\_scripts\_installed

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

1.5.0SVN

(1 row)

**相关请参考**

PostGIS\_Full\_Version, PostGIS\_Scripts\_Released, PostGIS\_Version



90 / 686

**8.3.12**

**PostGIS\_脚本发布函数**

PostGIS\_Scripts\_Released —返回postgis.sql脚本安装postgis库时候的脚本版本

**语法**

text PostGIS\_Scripts\_Released();

**描述**

返回postgis.sql脚本安装postgis库时候的脚本版本.

**注意**

从 1.1.0版本开始，该函数返回值和PostGIS\_Lib\_Version返回值一样，保持后向兼容

该函数可用版本: 0.9.0

**样例**

SELECT PostGIS\_Scripts\_Released();

postgis\_scripts\_released

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

1.3.4SVN

(1 row)

**相关请参考**

PostGIS\_Full\_Version, PostGIS\_Scripts\_Installed, PostGIS\_Lib\_Version

**8.3.13**

**PostGIS\_Version**

PostGIS\_Version —返回PostGIS的版本号和编译PostGIS时候的参数选项.

**语法**

text PostGIS\_Version();

**描述**

返回PostGIS的版本号和编译PostGIS时候的参数选项.

**样例**

SELECT PostGIS\_Version();

postgis\_version

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

1.3 USE\_GEOS=1 USE\_PROJ=1 USE\_STATS=1

(1 row)



91 / 686

**相关请参考**

PostGIS\_Full\_Version, PostGIS\_GEOS\_Version, PostGIS\_Lib\_Version, PostGIS\_LibXML\_Version, PostGIS\_PROJ\_Version

**8.3.14**

**Populate\_Geometry\_Columns**

Populate\_Geometry\_Columns — 确保geometry类型的列是用typmod方式来定义的或者有合适的空间约束条件，这样才能保证这些列能被正确地注册到geometry\_columns这个视图中。默认情况下会将所有的非typmod定义的geometry类型列转换为typmod方式的，如果想用老的方式，需要把use\_typmod值设为false

**语法**

text Populate\_Geometry\_Columns(boolean use\_typmod=true);

int Populate\_Geometry\_Columns(oid relation\_oid, boolean use\_typmod=true);

**描述**

确保geometry类型的列是用typmod方式来定义的或者有合适的空间约束条件，这样才能保证这些列能被正确地注册到geometry\_columns这个视图中

为了后向兼容性以及一些空间需求例如表继承，即每个指标可能有不同的geometry类型。老的约束方式依然支持。如果你想使用老的方式，你需要把use\_typmod设置为false.当传递这个参数后，geometry类型的列会被设置成无类型修改符的属性，但是会有3个约束条件，这意味着每个geometry类型的列在一个表中至少有3个约束（译者注：老的方式还是很复杂的，使用typmod方式吧，更符合建表习惯）：

· enforce\_dims\_the\_geom –确保每个geometry类型字段的维度相同(see ST\_NDims)

· enforce\_geotype\_the\_geom -确保每个geometry类型字段类型相同(see GeometryType)

· enforce\_srid\_the\_geom –确保每个geometry类型字段处于相同的投影空间(see ST\_SRID)

如果提供了一个表的oid，这个函数会试着确定srid，dimension值和该表的所有geometry类型，并添加必要的约束。如果成功了，一个正确的记录会被“插入”到geometry\_cloumns视图中(译者注：原文写的是插入geometry\_columns 表中，但实际上geometry\_columns是一个视图)，否则会导致一个异常并显示具体的错误信息

如果提供了一个视图的oid，就像一个表的oid一样，这个函数会试着确定srid，dimension值和该视图的所有geometry类型，如果成功了，一个正确的记录会被“插入”到geometry\_cloumns视图中，但是不会做任何事情来强制约束(译者注：如果是表的话会添加约束的)。

无参数方法（指的是返回值为text的第一种方法）是一个带参数的（返回值为int的）方法的封装，它首先对每一个空间类型表和视图先truncate geometry\_columns然后再重新插入到geometry\_columns，对每个表添加合适的空间约束，它返回在一个数据库能找到的所有geometry类型列的总和，和插入到geometry\_cloumns中的总数。参数化的版本简单返回插入到geometry\_columns中的总记录数

可用版本: 1.4.0

2.0.0版本更新：现在默认使用类型修改符而不是约束条件的方式来约束geometry类型的列，你依然可以通过设置use\_typmod为false的方式来使用老约束条件的方式

2.0.0版本更新：如果列是通过typmod方式或者使用约束的方式创建的，use\_typmod参数推荐使用

92 / 686

**样例**

CREATE TABLE public.myspatial\_table(gid serial, geom geometry);

INSERT INTO myspatial\_table(geom) VALUES(ST\_GeomFromText(‘LINESTRING(1 2, 3 4)’,4326) );

-- 这里会使用 type modifiers.

表中必须有数据才可以执行这个语句

SELECT Populate\_Geometry\_Columns(‘public.myspatial\_table’::regclass);

populate\_geometry\_columns

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

1

\d myspatial\_table

Table "public.myspatial\_table"

Column |

Type

|

Modifiers

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

gid

| integer

regclass)

| not null default nextval(‘myspatial\_table\_gid\_seq’::

geom

| geometry(LineString,4326) |

-- 如果没有用typmod或者已经有约束了，这里会改变geometry类型的列使用约束

--下面语句如果想要运行成功，表中需要有数据

CREATE TABLE public.myspatial\_table\_cs(gid serial, geom geometry);

INSERT INTO myspatial\_table\_cs(geom) VALUES(ST\_GeomFromText(‘LINESTRING(1 2, 3 4)’,4326) );

SELECT Populate\_Geometry\_Columns(‘public.myspatial\_table\_cs’::regclass, false);

populate\_geometry\_columns

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

1

\d myspatial\_table\_cs

Table "public.myspatial\_table\_cs"

Column |

Type

|

Modifiers

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

gid

| integer

| not null default nextval(‘myspatial\_table\_cs\_gid\_seq’::regclass)

geom

| geometry |

Check constraints:

"enforce\_dims\_geom" CHECK (st\_ndims(geom) = 2)

"enforce\_geotype\_geom" CHECK (geometrytype(geom) = ‘LINESTRING’::text OR geom IS NULL)

"enforce\_srid\_geom" CHECK (st\_srid(geom) = 4326)

**8.3.15**

**UpdateGeometrySRID**

UpdateGeometrySRID — 更新包括一个geometry类型列,该列在geometry\_columns 元数据信息及SRID的所有信息。如果该列被添加了强制约束，强制约束会被更新为新的SRID，如果是用typmod形式的定义，类型修改符也会被更新

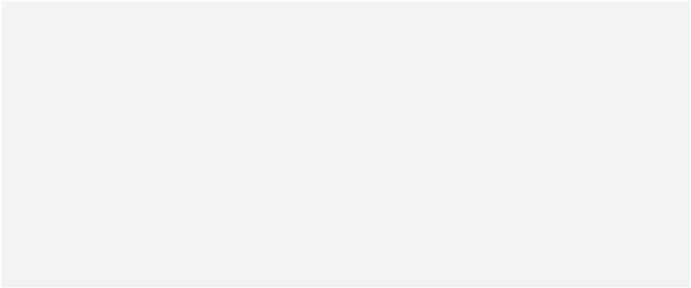
**语法**

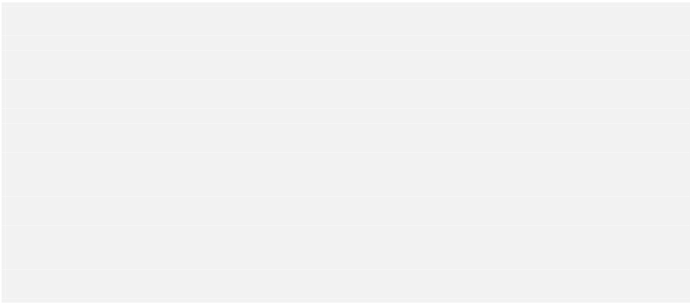
text UpdateGeometrySRID(varchar table\_name, varchar column\_name, integer srid);

text UpdateGeometrySRID(varchar schema\_name, varchar table\_name, varchar column\_name, integer srid);

text UpdateGeometrySRID(varchar catalog\_name, varchar schema\_name, varchar table\_name, varchar column\_name, integer

srid);





93 / 686

**描述**

更新包括一个geometry类型列,该列在geometry\_columns 元数据信息及SRID的所有信息. 注意：如果该函数没有设置schema\_name的值，将使用: current\_schema() 的值作为schema\_name(译者注：即select current\_schema()得到的值)

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

该方法支持CircularString和curve几何类型字段

**样例**

下面的语句将会把表roads的列geom的SRID改为4326，无论改列之前的SRI值是多少

SELECT UpdateGeometrySRID(‘roads’,’geom’,4326);

上面的样例语句等价于下面的DDL语句

ALTER TABLE roads

ALTER COLUMN geom TYPE geometry(MULTILINESTRING, 4326)

USING ST\_SetSRID(geom,4326);

如果你的数据在装载时候投影错了，你想一次性把投影坐标系改为Web Mercator投影，你可以使用下面的DDL语句，但是没有等价的PostGIS管理函数能一次性完成这样的事情(译者注：这句话的意思是DDL语句要更加强大)。

ALTER TABLE roads

ALTER COLUMN geom TYPE geometry(MULTILINESTRING, 3857) USING ST\_Transform(ST\_SetSRID(geom ,4326),3857) ;

**相关请参考**

ST\_SetSRID , ST\_Transform

**8.4**

**8.4.1**

**几何对象创建函数**

**ST\_BdPolyFromText**

ST\_BdPolyFromText — 根据一个任意的封闭的WKT描述的MultiLineString几何类型对象创建一个Polygon对象

**语法**

geometry ST\_BdPolyFromText(text WKT, integer srid);

**描述**

根据一个任意的封闭的WKT描述的MultiLineString几何类型对象创建一个Polygon对象.

**注意**

如果WKT描述的对象并不是MULTILINESTRING类型的对象，会抛出异常.如果输出是MULTIPOLYGON也会抛出异常; 遇到这种情况使用ST\_BdMPolyFromText函数,或者查看 ST\_BuildArea() 寻找postgis专有的办法

该方法实现了OpenGIS 关于SQL 1.1的简单特性实现规范OpenGIS Simple Features Implementation Speciﬁcation for SQL 1.1. s3.2.6.2

Availability: 1.1.0 - requires GEOS >= 2.1.0.









94 / 686

**样例**

Forthcoming（即将给出译者注：原文就没有给出，卧槽。。还带这样的）

**相关请参考**

ST\_BuildArea, ST\_BdMPolyFromText

**8.4.2**

**ST\_BdMPolyFromText**

ST\_BdMPolyFromText — 根据一个任意的封闭的WKT描述的MultiLineString几何类型对象创建一个MultiPolygon对象.

**语法**

geometry ST\_BdMPolyFromText(text WKT, integer srid);

**描述**

根据一个任意的封闭的WKT描述的MultiLineString几何类型对象创建一个MultiPolygon对象

**注意**

如果WKT描述的对象并不是MULTILINESTRING类型的对象. 即使结果真的只有一个多边形组成，也会强制输出MULTIPOLYGON对象，遇到这种情况，如果你确定输出是POLYGON对象，请使用ST\_BdPolyFromText，或者查看 ST\_BuildArea() 寻找postgis专有的办法

该方法实现了OpenGIS 关于SQL 1.1的简单特性实现规范OpenGIS Simple Features Implementation Speciﬁcation for SQL 1.1. s3.2.6.2

Availability: 1.1.0 - requires GEOS >= 2.1.0.

**样例**

本文档么得，即将给出

**相关请参考**

ST\_BuildArea, ST\_BdPolyFromText

**8.4.3**

**ST\_Box2dFromGeoHash**

ST\_Box2dFromGeoHash — 从一个GeoHash字符串返回一个BOX2D对象

**语法**

box2d ST\_Box2dFromGeoHash(text geohash, integer precision=full\_precision\_of\_geohash);





95 / 686

**描述**

从一个GeoHash字符串返回一个BOX2D对象.

如果precision参数没有给出，ST\_Box2dFromGeoHash函数根据输入的GeoHash字符串的全精度返回一个BOX2D对象。如果指定了precision参数，ST\_Box2dFromGeoHash函数会从GeoHash字符串中使用很多字符来创建一个BOX2D对象。更低的精度值会产生更大的BOX2D对象，而更大的精度值会提高精度。

可用版本: 2.1.0

**样例**

SELECT ST\_Box2dFromGeoHash(‘9qqj7nmxncgyy4d0dbxqz0’);

st\_geomfromgeohash

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

BOX(-115.172816 36.114646,-115.172816 36.114646)

SELECT ST\_Box2dFromGeoHash(‘9qqj7nmxncgyy4d0dbxqz0’, 0);

st\_box2dfromgeohash

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

BOX(-180 -90,180 90)

SELECT ST\_Box2dFromGeoHash(‘9qqj7nmxncgyy4d0dbxqz0’, 10);

st\_box2dfromgeohash

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

BOX(-115.17282128334 36.1146408319473,-115.172810554504 36.1146461963654)

**相关请参考**

ST\_GeoHash, ST\_GeomFromGeoHash, ST\_PointFromGeoHash

**8.4.4**

**ST\_GeogFromText**

ST\_GeogFromText —从一个WKT规范描述的对象返回一个具体的geography对象

**语法**

geography ST\_GeogFromText(text EWKT);

**描述**

从一个WKT或EWKT规范描述的对象返回一个具体的geography对象.。SRID会默认设置为4326.这个函数是

ST\_GeographyFromText的别名（即两个函数其实是一样的）. Point类型必须用经纬度的形式做参数。

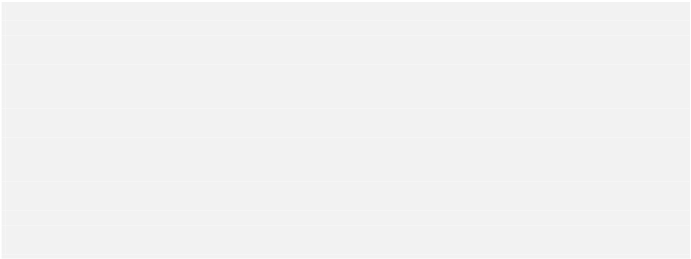
**样例**

--- 把经纬度坐标转换为geography坐标

ALTER TABLE sometable ADD COLUMN geog geography(POINT,4326);

UPDATE sometable SET geog = ST\_GeogFromText(‘SRID=4326;POINT(‘ || lon || ‘ ‘ || lat || ‘)’)

;



96 / 686

**相关请参考**

ST\_AsText, ST\_GeographyFromText

**8.4.5**

**ST\_GeographyFromText**

ST\_GeographyFromText —从一个WKT规范描述的对象返回一个具体的geography对象.

**语法**

geography ST\_GeographyFromText(text EWKT);

**描述**

从一个WKT或EWKT规范描述的对象返回一个具体的geography对象.。SRID会默认设置为4326.

**相关请参考**

ST\_GeogFromText, ST\_AsText

**8.4.6**

**ST\_GeogFromWKB**

ST\_GeogFromWKB —从一个WKB或EWKB规范描述的对象返回一个具体的geography对象

**语法**

geography ST\_GeogFromWKB(bytea geom);

**描述**

从一个WKB或EWKB规范描述的对象返回一个具体的geography对象. 该函数在SQL中充当几何类型工厂的角色，如果SRID没有指定，函数会默认设置为4326(即WGS 84 long lat坐标系)

该方法支持CircularString和curve几何类型字段

**样例**

--虽然bytea 描述包含单斜杠\，但当插入数据时候，这些符号需要使用逃逸字符E处理掉

SELECT ST\_AsText(

ST\_GeogFromWKB(E’\\001\\002\\000\\000\\000\\002\\000\\000\\000\\037\\205\\353Q

\\270~\\\\\\300\\323Mb\\020X\\231C@\\020X9\\264\\310~\\\\\\300)\\\\\\217\\302\\365\\230

C@’)

);

st\_astext

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

LINESTRING(-113.98 39.198,-113.981 39.195)

(1 row)



97 / 686

**相关请参考**

ST\_GeogFromText, ST\_AsBinary

**8.4.7**

**ST\_GeomCollFromText**

ST\_GeomCollFromText — 根据WKT和给定的SRID值创建一个geometry collection类型的对象，如果SRID值没有给出，默认为0

**语法**

geometry ST\_GeomCollFromText(text WKT, integer srid);

geometry ST\_GeomCollFromText(text WKT);

**描述**

根据WKT和给定的SRID值创建一个geometry collection类型的对象，如果SRID值没有给出，默认为0.

OGC SPEC 3.2.6.2 - 参数SRID需要满足该规范

如果WKT描述不是一个GEOMETRYCOLLECTION对象，返回null

**注意**

如果你绝对确定你输入的WKT geometry类型是collection类型的，不要使用这个函数，它会比ST\_GeomFromText 这个函数慢得多，因为ST\_GeomCollFromText这个函数会进行额外的类型校验步骤。

该方法实现了OpenGIS 关于SQL 1.1的简单特性实现规范OpenGIS Simple Features Implementation Speciﬁcation for SQL 1.1. s3.2.6.2

该函数实现了SQL/MM 标准.

**样例**

SELECT ST\_GeomCollFromText(‘GEOMETRYCOLLECTION(POINT(1 2),LINESTRING(1 2, 3 4))’);

**相关请参考**

ST\_GeomFromText, ST\_SRID

**8.4.8**

**ST\_GeomFromEWKB**

ST\_GeomFromEWKB —从一个EWKB描述的几何对象返回一个具体的ST\_Geometry值.

**语法**

geometry ST\_GeomFromEWKB(bytea EWKB);







98 / 686

**描述**

从一个OGC EWKB标准的描述的几何对象返回一个具体的PostGIS ST\_Geometry对象

**注意**

EWKB 格式不是OGC标准，而是PostGIS的具体的一种包含SRID的格式

提升: 2.0.0版本支持Polyhedral Surface类型和TIN类型

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

该方法支持CircularString和curve几何类型字段

该函数支持Polyhedral Surface类型.

该函数支持 Triangles 和 Triangulated Irregular Network Surfaces (TIN)类型.

**样例**

line string binary rep 0f LINESTRING(-71.160281 42.258729,-71.160837 42.259113,-71.161144 42.25932) in NAD 83 long lat

(4269).

**注意**

注意: 即便 byte类型数组是用\分隔的，也许还有单引号‘，如果standard\_conforming\_strings值为off(译者注：standard\_conforming\_strings是PostgreSQL的一个参数)，我们需要用反斜线\和双引号"来进行转义。因此这个参数看起来不像它应该是EWKB方式描述的参数

SELECT ST\_GeomFromEWKB(E’\\001\\002\\000\\000 \\255\\020\\000\\000\\003\\000\\000\\000\\344

J=

\\013B\\312Q\\300n\\303(\\010\\036!E@’’\\277E’’K

\\312Q\\300\\366{b\\235\*!E@\\225|\\354.P\\312Q

\\300p\\231\\323e1!E@’);

**注意**

在 PostgreSQL 9.1+ 及更高版本-standard\_conforming\_strings 默认值是on, 和之前的版本一样。你可以根据需要对单个查询或者对全局的数据库修改该值。下面将告诉你怎样在设置.standard\_conforming\_strings值为on的情况下，使用该函数。在这里，我们使用标准的ansi编码的单引号'，对引号'进行转义，但是斜线并没有被转义

set standard\_conforming\_strings = on;

SELECT ST\_GeomFromEWKB(‘\001\002\000\000 \255\020\000\000\003\000\000\000\344J=\012\013B

\312Q\300n\303(\010\036!E@’’\277E’’K\012\312Q\300\366{b\235\*!E@\225|\354.P\312Q\012\300

p\231\323e1’)

**相关请参考**

ST\_AsBinary, ST\_AsEWKB, ST\_GeomFromWKB















99 / 686

**8.4.9**

**ST\_GeomFromEWKT**

ST\_GeomFromEWKT — 从一个EWKT描述的几何对象返回一个具体的ST\_Geometry值.

**语法**

geometry ST\_GeomFromEWKT(text EWKT);

**描述**

从一个OGC EWKT标准的描述的几何对象返回一个具体的PostGIS ST\_Geometry对象

**注意**

EWKT格式不是OGC标准，而是PostGIS的具体的一种包含SRID的格式

提升: 提升: 2.0.0版本支持Polyhedral Surface类型和TIN类型

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

该方法支持CircularString和curve几何类型字段

该函数支持Polyhedral Surface类型.

该函数支持 Triangles 和 Triangulated Irregular Network Surfaces (TIN)类型.

**样例**

SELECT ST\_GeomFromEWKT(‘SRID=4269;LINESTRING(-71.160281 42.258729,-71.160837

42.259113,-71.161144 42.25932)’);

SELECT ST\_GeomFromEWKT(‘SRID=4269;MULTILINESTRING((-71.160281 42.258729,-71.160837

42.259113,-71.161144 42.25932))’);

SELECT ST\_GeomFromEWKT(‘SRID=4269;POINT(-71.064544 42.28787)’);

SELECT ST\_GeomFromEWKT(‘SRID=4269;POLYGON((-71.1776585052917

42.3902909739571,-71.1776820268866 42.3903701743239,

-71.1776063012595 42.3903825660754,-71.1775826583081 42.3903033653531,-71.1776585052917

42.3902909739571))’);

SELECT ST\_GeomFromEWKT(‘SRID=4269;MULTIPOLYGON(((-71.1031880899493 42.3152774590236,

-71.1031627617667 42.3152960829043,-71.102923838298 42.3149156848307,

-71.1023097974109 42.3151969047397,-71.1019285062273 42.3147384934248,

-71.102505233663 42.3144722937587,-71.10277487471 42.3141658254797,

-71.103113945163 42.3142739188902,-71.10324876416 42.31402489987,

-71.1033002961013 42.3140393340215,-71.1033488797549 42.3139495090772,

-71.103396240451 42.3138632439557,-71.1041521907712 42.3141153348029,

-71.1041411411543 42.3141545014533,-71.1041287795912 42.3142114839058,

-71.1041188134329 42.3142693656241,-71.1041112482575 42.3143272556118,

-71.1041072845732 42.3143851580048,-71.1041057218871 42.3144430686681,

-71.1041065602059 42.3145009876017,-71.1041097995362 42.3145589148055,

-71.1041166403905 42.3146168544148,-71.1041258822717 42.3146748022936,

-71.1041375307579 42.3147318674446,-71.1041492906949 42.3147711126569,

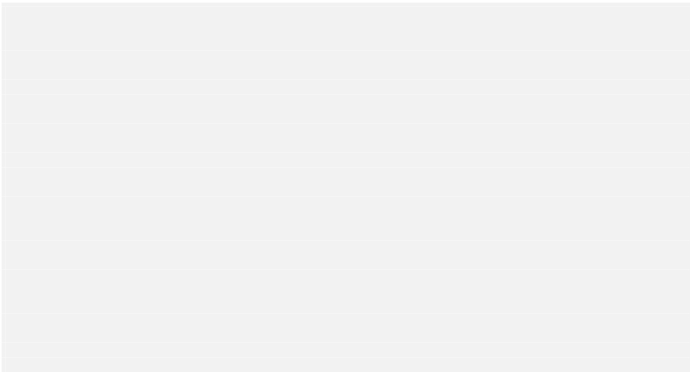












100 / 686

-71.1041598612795 42.314808571739,-71.1042515013869 42.3151287620809,

-71.1041173835118 42.3150739481917,-71.1040809891419 42.3151344119048,

-71.1040438678912 42.3151191367447,-71.1040194562988 42.3151832057859,

-71.1038734225584 42.3151140942995,-71.1038446938243 42.3151006300338,

-71.1038315271889 42.315094347535,-71.1037393329282 42.315054824985,

-71.1035447555574 42.3152608696313,-71.1033436658644 42.3151648370544,

-71.1032580383161 42.3152269126061,-71.103223066939 42.3152517403219,

-71.1031880899493 42.3152774590236)),

((-71.1043632495873 42.315113108546,-71.1043583974082 42.3151211109857,

-71.1043443253471 42.3150676015829,-71.1043850704575 42.3150793250568,-71.1043632495873

42.315113108546)))’);

--3d circular string

SELECT ST\_GeomFromEWKT(‘CIRCULARSTRING(220268 150415 1,220227 150505 2,220227 150406 3)’);

--Polyhedral Surface example

SELECT ST\_GeomFromEWKT(‘POLYHEDRALSURFACE(

((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)),

((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)),

((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),

((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),

((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)),

((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1))

)’);

**相关请参考**

ST\_AsEWKT, ST\_GeomFromText, ST\_GeomFromEWKT

**8.4.10**

**ST\_GeometryFromText**

ST\_GeometryFromText — 根据WKT描述的对象返回一个具体的ST\_Geometry 函数值(也是一个geometry对象)，该函数是ST\_GeomFromText的别名，即两者等价

**语法**

geometry ST\_GeometryFromText(text WKT);

geometry ST\_GeometryFromText(text WKT, integer srid);

**描述**

该方法实现了OpenGIS 关于SQL 1.1的简单特性实现规范 [OpenGIS Simple Features Implementation Speciﬁcation for SQL 1.1.](http://www.opengeospatial.org/standards/sfs)

该方法实现了 SQL/MM 标准版本SQL-MM 3: 5.1.40

**相关请参考**

ST\_GeomFromText

**8.4.11**

**ST\_GeomFromGeoHash**

ST\_GeomFromGeoHash —从一个GeoHash型字符串返回一个geometry对象类型.





101 / 686

**语法**

geometry ST\_GeomFromGeoHash(text geohash, integer precision=full\_precision\_of\_geohash);

**描述**

从一个GeoHash字符串返回一个geometry类型对象，该geometry对象会是一个polygon对象，表示GeoHash的边界

如果precision参数没有给出，ST\_GeomFromGeoHash函数根据输入的GeoHash字符串的全精度返回一个polygon对象。如果指定了precision参数，ST\_GeomFromGeoHash函数会从GeoHash字符串中使用很多字符来创建一个polygon对象。

可用版本: 2.1.0

**样例**

SELECT ST\_AsText(ST\_GeomFromGeoHash(‘9qqj7nmxncgyy4d0dbxqz0’));

st\_astext

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

POLYGON((-115.172816 36.114646,-115.172816 36.114646,-115.172816 36.114646,-115.172816

36.114646,-115.172816 36.114646))

SELECT ST\_AsText(ST\_GeomFromGeoHash(‘9qqj7nmxncgyy4d0dbxqz0’, 4));

st\_astext

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

POLYGON((-115.3125 36.03515625,-115.3125 36.2109375,-114.9609375 36.2109375,-114.9609375

36.03515625,-115.3125 36.03515625))

SELECT ST\_AsText(ST\_GeomFromGeoHash(‘9qqj7nmxncgyy4d0dbxqz0’, 10));

st\_astext

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

POLYGON((-115.17282128334 36.1146408319473,-115.17282128334

36.1146461963654,-115.172810554504 36.1146461963654,-115.172810554504

36.1146408319473,-115.17282128334 36.1146408319473))

**相关请参考**

ST\_GeoHash,ST\_Box2dFromGeoHash, ST\_PointFromGeoHash

**8.4.12**

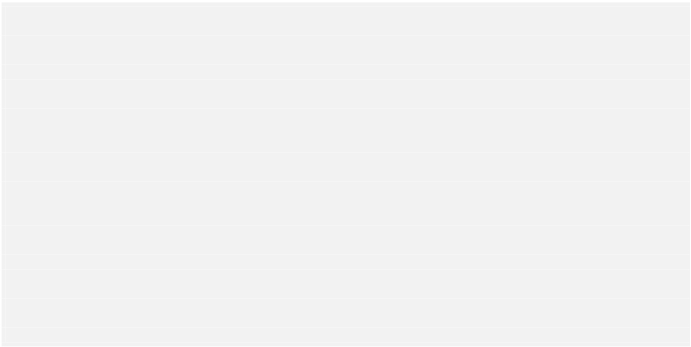
**ST\_GeomFromGML**

ST\_GeomFromGML —根据一个geometry对象的GML描述，返回一个PostGIS的geometry类型对象

**语法**

geometry ST\_GeomFromGML(text geomgml);

geometry ST\_GeomFromGML(text geomgml, integer srid);



102 / 686

**描述**

根据一个geometry对象的OGC GML描述，返回一个PostGIS的geometry类型对象.

ST\_GeomFromGML 只对GML描述的Geometry fragment元素标记类型输入有效，如果你试着在整个GML文档上执行该函数，将会抛出异常。

OGC GML 支持下面的版本:

· GML 3.2.1 Namespace

· GML 3.1.1 Simple Features proﬁle SF-2 (GML 3.1.0 and 3.0.0 后向兼容)

· GML 2.1.2

OGC GML 规范, 参考: [http://www.opengeospatial.org/standards/gml:](http://www.opengeospatial.org/standards/gml)

可用版本: 1.5, 需要 libxml2 1.6+版本及更高版本支持

提升: 提升: 2.0.0版本支持Polyhedral Surface类型和TIN类型

提升: 2.0.0 添加SRID参数选项.

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

该函数支持Polyhedral Surface类型.

该函数支持 Triangles 和 Triangulated Irregular Network Surfaces (TIN)类型.

GML 允许混合维度(例如在一个MultiGeometry类型中同时包含2D和3D对象). 而PostGIS 不允许这样，所以函数

ST\_GeomFromGML 一旦发现一个Z维度值缺失，就会把整个geometry对象转换为2D对象。

GML支持在同一个MultiGeometry类型中支持混合SRS对象，而 PostGIS 几何对象不允许这样, ST\_GeomFromGML,在这种情下会

将整个子几何类型投影到SRS根节点类型。如果GML根节点没有可用的srsName值，这个函数会抛出一个异常。

ST\_GeomFromGML函数并关于GML namespace并非迂腐的做法。常规使用中，你可以避免明确提及它（指GML namespace），

但是如果你想在GML中使用XLink特性，你需要它

**注意**

ST\_GeomFromGML 函数不支持 SQL/MM curve几何类型对象.

**样例 - 一个带有 srsName属性值的几何对象**

SELECT ST\_GeomFromGML(‘

<gml:LineString srsName="EPSG:4269">

<gml:coordinates>

-71.16028,42.258729 -71.160837,42.259112 -71.161143,42.25932

</gml:coordinates>

</gml:LineString>‘);

**样例 - XLink 使用**









103 / 686

SELECT ST\_GeomFromGML(‘

<gml:LineString xmlns:gml="http://www.opengis.net/gml"

xmlns:xlink="http://www.w3.org/1999/xlink"

srsName="urn:ogc:def:crs:EPSG::4269">

<gml:pointProperty>

<gml:Point gml:id="p1"><gml:pos>42.258729 -71.16028</gml:pos></gml:Point>

</gml:pointProperty>

<gml:pos>42.259112 -71.160837</gml:pos>

<gml:pointProperty>

<gml:Point xlink:type="simple" xlink:href="#p1"/>

</gml:pointProperty>

</gml:LineString>‘););

**样例 - Polyhedral Surface**

SELECT ST\_AsEWKT(ST\_GeomFromGML(‘

<gml:PolyhedralSurface>

<gml:polygonPatches>

<gml:PolygonPatch>

<gml:exterior>

<gml:LinearRing><gml:posList srsDimension="3">0

posList></gml:LinearRing>

</gml:exterior>

</gml:PolygonPatch>

<gml:PolygonPatch>

<gml:exterior>

<gml:LinearRing><gml:posList srsDimension="3">0

posList></gml:LinearRing>

</gml:exterior>

</gml:PolygonPatch>

<gml:PolygonPatch>

<gml:exterior>

<gml:LinearRing><gml:posList srsDimension="3">0

posList></gml:LinearRing>

</gml:exterior>

</gml:PolygonPatch>

<gml:PolygonPatch>

<gml:exterior>

<gml:LinearRing><gml:posList srsDimension="3">1

posList></gml:LinearRing>

</gml:exterior>

</gml:PolygonPatch>

<gml:PolygonPatch>

<gml:exterior>

<gml:LinearRing><gml:posList srsDimension="3">0

posList></gml:LinearRing>

</gml:exterior>

</gml:PolygonPatch>

<gml:PolygonPatch>

<gml:exterior>

<gml:LinearRing><gml:posList srsDimension="3">0

posList></gml:LinearRing>

</gml:exterior>

</gml:PolygonPatch>

</gml:polygonPatches>

</gml:PolyhedralSurface>‘));

-- result --

POLYHEDRALSURFACE(((0 0 0,0 0 1,0 1 1,0 1 0,0 0 0)),

0 0 0 0 1 0 1 1 0 1 0 0 0 0</gml:

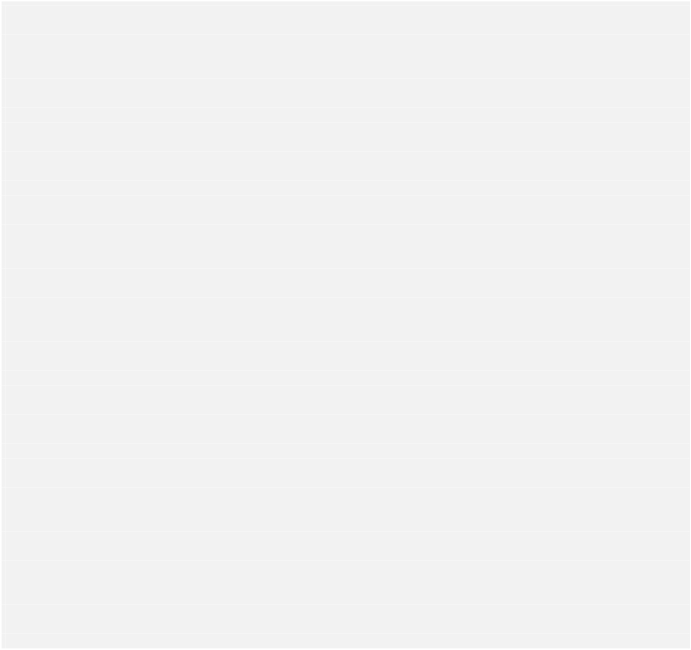
0 0 0 1 0 1 1 0 1 0 0 0 0 0</gml:

0 0 1 0 0 1 0 1 0 0 1 0 0 0</gml:

1 0 1 1 1 1 0 1 1 0 0 1 1 0</gml:

1 0 0 1 1 1 1 1 1 1 0 0 1 0</gml:

0 1 1 0 1 1 1 1 0 1 1 0 0 1</gml:



104 / 686

((0

((0

((1

((0

((0

0

0

1

1

0

0,0

0,1

0,1

0,0

1,1

1

0

1

1

0

0,1

0,1

1,1

1,1

1,1

1

0

0

1

1

0,1

1,0

1,1

1,1

1,0

0

0

0

1

1

0,0

1,0

0,1

0,0

1,0

0

0

1

1

0

0)),

0)),

0)),

0)),

1)))

**相关请参考**

Section 2.4.1, ST\_AsGML, ST\_GMLToSQL

**8.4.13**

**ST\_GeomFromGeoJSON**

ST\_GeomFromGeoJSON —该函数根据一个geojson描述的几何对象，生成一个PostGIS 的geometry对象

**语法**

geometry ST\_GeomFromGeoJSON(text geomjson);

**描述**

该函数根据一个geojson描述的几何对象，生成一个PostGIS 的geometry对象

ST\_GeomFromGeoJSON 只能在Json 的Geometry的元素标记起作用，如果你把整个JSON文档当做该函数的输入，将会抛出异常

可用版本: 2.0.0 需要 - JSON-C版本0.9及以上版本支持

**注意**

如果你没有添加JSON-C支持，也许你会得到一个错误提示，而不是一个正常的输出。要想添加JSON-C支持，需要运行：

run conﬁgure --with-jsondir=/path/to/json-c. 查看 Section 2.4.1获取更多细节.

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

**样例**

SELECT ST\_AsText(ST\_GeomFromGeoJSON(‘{"type":"Point","coordinates":[-48.23456,20.12345]}’))

As wkt;

wkt

------

POINT(-48.23456 20.12345)

-- a 3D linestring

SELECT ST\_AsText(ST\_GeomFromGeoJSON(‘{"type":"LineString","coordinates

":[[1,2,3],[4,5,6],[7,8,9]]}’)) As wkt;

wkt

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

LINESTRING(1 2,4 5,7 8)





105 / 686

**相关请参考**

ST\_AsText, ST\_AsGeoJSON, Section 2.4.1

**8.4.14**

**ST\_GeomFromKML**

ST\_GeomFromKML —该函数根据一个KML描述的几何对象，生成一个PostGIS 的geometry对象

**语法**

geometry ST\_GeomFromKML(text geomkml);

**描述**

该函数根据一个OGC KML描述的几何对象，生成一个PostGIS 的geometry对象

ST\_GeomFromKML只能在KML的Geometry的元素标记起作用，如果你把整个KML文档当做该函数的输入，将会抛出异常

OGC KML 如下版本支持:

· KML 2.2.0 Namespace

OGC KML 规范, 参考: [http://www.opengeospatial.org/standards/kml:](http://www.opengeospatial.org/standards/kml)

可用版本: 1.5,libxml2 需要版本2.6+

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

**注意**

ST\_GeomFromKML 函数不支持SQL/MM curve几何类型.

**样例 - A single geometry with srsName**

SELECT ST\_GeomFromKML(‘

<LineString>

<coordinates>-71.1663,42.2614

-71.1667,42.2616</coordinates>

</LineString>‘);

**相关请参考**

Section 2.4.1, ST\_AsKML

**8.4.15**

**ST\_GMLToSQL**

ST\_GMLToSQL — 根据GML表述的对象返回一个ST\_Geometry值，这个函数是ST\_GeomFromGML的别名

**语法**

geometry ST\_GMLToSQL(text geomgml);

geometry ST\_GMLToSQL(text geomgml, integer srid);





106 / 686

**描述**

该方法实现了 SQL/MM speciﬁcation. SQL-MM 3: 5.1.50规范 (但不包括对curve类型的支持).

可用版本: 1.5, requires libxml2 1.6+

提升: 2.0.0版本支持Polyhedral Surface类型和TIN类型

提升: 2.0.0 添加了SRID参数选项.

**相关请参考**

Section 2.4.1, ST\_GeomFromGML, ST\_AsGML

**8.4.16**

**ST\_GeomFromText**

ST\_GeomFromText — 根据WKT描述返回一个具体的ST\_Geometry 值

**语法**

geometry ST\_GeomFromText(text WKT);

geometry ST\_GeomFromText(text WKT, integer srid);

**描述**

从OGC Well-Known格式文本描述创建一个PostGIS ST\_Geometry对象.

**注意**

ST\_GeomFromText 函数有两种变体(使用方式)，第一种没有SRID参数，返回的geometry类型没有定义空间参考系，第二种定义了第二个参数SRID，返回的geometry对象包含了SRID值作为元数据信息，SRID值必须是定义在表spatial\_ref\_sys中的一种

该方法实现了OpenGIS 关于SQL 1.1的简单特性实现规范OpenGIS Simple Features Implementation Speciﬁcation for SQL 1.1. s3.2.6.2 –可选参数SRID是为了满足该规范

该方法实现了 SQL/MM 标准： SQL-MM 3: 5.1.40

该方法支持CircularString和curve几何类型字段

**注意**

: 2.0.0版本更新：在早先的PostGIS版本，ST\_GeomFromText(‘GEOMETRYCOLLECTION(EMPTY)’) 允许执行，而在

PostGIS 2.0.0 版本为了遵守SQL/MM规范，这个写法是非法的。现在应该这样写

ST\_GeomFromText(‘GEOMETRYCOLLECTION EMPTY’)













107 / 686

**样例**

SELECT ST\_GeomFromText(‘LINESTRING(-71.160281 42.258729,-71.160837 42.259113,-71.161144

42.25932)’);

SELECT ST\_GeomFromText(‘LINESTRING(-71.160281 42.258729,-71.160837 42.259113,-71.161144

42.25932)’,4269);

SELECT ST\_GeomFromText(‘MULTILINESTRING((-71.160281 42.258729,-71.160837

42.259113,-71.161144 42.25932))’);

SELECT ST\_GeomFromText(‘POINT(-71.064544 42.28787)’);

SELECT ST\_GeomFromText(‘POLYGON((-71.1776585052917 42.3902909739571,-71.1776820268866

42.3903701743239,

-71.1776063012595 42.3903825660754,-71.1775826583081 42.3903033653531,-71.1776585052917

42.3902909739571))’);

SELECT ST\_GeomFromText(‘MULTIPOLYGON(((-71.1031880899493 42.3152774590236,

-71.1031627617667 42.3152960829043,-71.102923838298 42.3149156848307,

-71.1023097974109 42.3151969047397,-71.1019285062273 42.3147384934248,

-71.102505233663 42.3144722937587,-71.10277487471 42.3141658254797,

-71.103113945163 42.3142739188902,-71.10324876416 42.31402489987,

-71.1033002961013 42.3140393340215,-71.1033488797549 42.3139495090772,

-71.103396240451 42.3138632439557,-71.1041521907712 42.3141153348029,

-71.1041411411543 42.3141545014533,-71.1041287795912 42.3142114839058,

-71.1041188134329 42.3142693656241,-71.1041112482575 42.3143272556118,

-71.1041072845732 42.3143851580048,-71.1041057218871 42.3144430686681,

-71.1041065602059 42.3145009876017,-71.1041097995362 42.3145589148055,

-71.1041166403905 42.3146168544148,-71.1041258822717 42.3146748022936,

-71.1041375307579 42.3147318674446,-71.1041492906949 42.3147711126569,

-71.1041598612795 42.314808571739,-71.1042515013869 42.3151287620809,

-71.1041173835118 42.3150739481917,-71.1040809891419 42.3151344119048,

-71.1040438678912 42.3151191367447,-71.1040194562988 42.3151832057859,

-71.1038734225584 42.3151140942995,-71.1038446938243 42.3151006300338,

-71.1038315271889 42.315094347535,-71.1037393329282 42.315054824985,

-71.1035447555574 42.3152608696313,-71.1033436658644 42.3151648370544,

-71.1032580383161 42.3152269126061,-71.103223066939 42.3152517403219,

-71.1031880899493 42.3152774590236)),

((-71.1043632495873 42.315113108546,-71.1043583974082 42.3151211109857,

-71.1043443253471 42.3150676015829,-71.1043850704575 42.3150793250568,-71.1043632495873

42.315113108546)))’,4326);

SELECT ST\_GeomFromText(‘CIRCULARSTRING(220268 150415,220227 150505,220227 150406)’);

**相关请参考**

ST\_GeomFromEWKT, ST\_GeomFromWKB, ST\_SRID

**8.4.17**

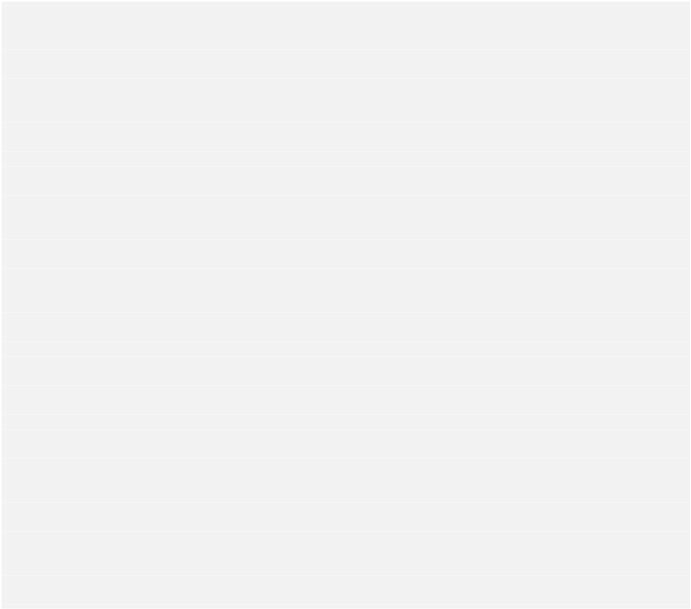
**ST\_GeomFromWKB**

ST\_GeomFromWKB — 根据WKB表述创建一个geometry几何类型对象，SRID可选

**语法**

geometry ST\_GeomFromWKB(bytea geom);

geometry ST\_GeomFromWKB(bytea geom, integer srid);



108 / 686

**描述**

根据WKB表述创建一个geometry几何类型对象，SRID可选，这个函数在SQL中充当Geometry Factory 的角色，该函数

ST\_WKBToSQL的候选别名

如果SRID没有指定，SRID将被设置为0，表示SRID未知

该方法实现了OpenGIS 关于SQL 1.1的简单特性实现规范OpenGIS Simple Features Implementation Speciﬁcation for SQL 1.1. s3.2.7.2 –可选参数SRID需要满足该规范­

该方法实现了 SQL/MM 标准： SQL-MM 3: 5.1.41

该方法支持CircularString和curve几何类型字段

**样例**

--Although bytea rep contains single \, these need to be escaped when inserting into a

table

-- unless standard\_conforming\_strings is set to on.

SELECT ST\_AsEWKT(

ST\_GeomFromWKB(E’\\001\\002\\000\\000\\000\\002\\000\\000\\000\\037\\205\\353Q

\\270~\\\\\\300\\323Mb\\020X\\231C@\\020X9\\264\\310~\\\\\\300)\\\\\\217\\302\\365\\230

C@’,4326)

);

st\_asewkt

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

SRID=4326;LINESTRING(-113.98 39.198,-113.981 39.195)

(1 row)

SELECT

ST\_AsText(

ST\_GeomFromWKB(

ST\_AsEWKB(‘POINT(2 5)’::geometry)

)

);

st\_astext

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

POINT(2 5)

(1 row)

**相关请参考**

ST\_WKBToSQL, ST\_AsBinary, ST\_GeomFromEWKB

**8.4.18**

**ST\_LineFromMultiPoint**

ST\_LineFromMultiPoint —从一个MultiPoint几何类型中返回一个LineString类型对象.

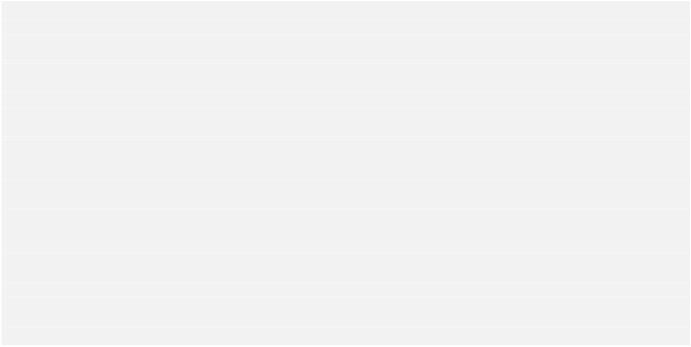
**语法**

geometry ST\_LineFromMultiPoint(geometry aMultiPoint)









PostGIS 2.2.0dev开发手册

109 / 686

**描述**

用MultiPoint几何类型对象创建一个LineString 类型对象.

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

**样例**

--Create a 3d line string from a 3d multipoint

SELECT ST\_AsEWKT(ST\_LineFromMultiPoint(ST\_GeomFromEWKT(’MULTIPOINT(1 2 3, 4 5 6, 7 8 9)’))) ←

;

--result--

LINESTRING(1 2 3,4 5 6,7 8 9)

**参考**

ST\_AsEWKT, ST\_Collect, ST\_MakeLine

**8.4.19**

**ST\_LineFromText**

ST\_LineFromText —根据WKT表述和给定的SRID创建一个geometry几何类型对象，如果SRID没有给定默认设置为0.

**用法**

geometry ST\_LineFromText(text WKT);

geometry ST\_LineFromText(text WKT, integer srid);

**描述**

根据WKT表述和给定的SRID创建一个geometry几何类型对象，如果SRID没有给定默认设置为0. 如果传递的WKT参数不是LINESTRING返回值将为空

**注意**

OGC SPEC 3.2.6.2 – SRID为了满足规范，保持和该规范一致性

**注意：**

如果你十分确定你的几何类型列是LINESTRING,使用ST\_GeomFromText性能更高. 而ST\_LineFromText函数只是调用ST\_GeomFromText 这个函数，同时添加了对LINESTRING类型的校验

该方法实现了规范 OpenGIS Simple Features Implementation Speciﬁcation for SQL 1.1. s3.2.6.2

该方法实现了规范 SQL/MM speciﬁcation. SQL-MM 3: 7.2.8











PostGIS 2.2.0dev开发手册

110 / 686

**样例**

SELECT ST\_LineFromText(’LINESTRING(1 2, 3 4)’) AS aline, ST\_LineFromText(’POINT(1 2)’) AS

null\_return;

←

aline

| null\_return

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

010200000002000000000000000000F ... | t

**参考**

ST\_GeomFromText

**8.4.20**

**ST\_LineFromWKB**

ST\_LineFromWKB —根据WKB表述和给定的SRID创建一个LINESTRING几何类型对象

**用法**

geometry ST\_LineFromWKB(bytea WKB);

geometry ST\_LineFromWKB(bytea WKB, integer srid);

**描述**

根据WKB表述和给定的SRID创建一个LINESTRING几何类型对象，该函数在SQL中充当Geometry Factory角色

如果SRID没有给定默认设置为0，如果输入的的bytea类型参数不是LINESTRING 几何类型对象，返回NULL值

**注意**

OGC SPEC 3.2.6.2 –参数SRID为了满足该规范一致性要求.

**注意**

如果你十分确定你的几何类型列是LINESTRING,使用ST\_GeomFromWKB性能更高. 而ST\_LineFromWKB函数只是调用ST\_GeomFromWKB这个函数，同时添加了对LINESTRING类型的校验

该方法实现了规范 OpenGIS Simple Features Implementation Speciﬁcation for SQL 1.1. s3.2.6.2

该方法实现了规范 SQL/MM speciﬁcation. SQL-MM 3: 7.2.9

**样例**

SELECT ST\_LineFromWKB(ST\_AsBinary(ST\_GeomFromText(’LINESTRING(1 2, 3 4)’))) AS aline,

ST\_LineFromWKB(ST\_AsBinary(ST\_GeomFromText(’POINT(1 2)’))) IS NULL AS null\_return;

aline

| null\_return

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

010200000002000000000000000000F ... | t









PostGIS 2.2.0dev开发手册

111 / 686

**参考**

ST\_GeomFromWKB, ST\_LinestringFromWKB

**8.4.21**

**ST\_LinestringFromWKB**

ST\_LinestringFromWKB —根据WKB表述和给定的SRID创建一个geometry几何类型对象.

**用法**

geometry ST\_LinestringFromWKB(bytea WKB);

geometry ST\_LinestringFromWKB(bytea WKB, integer srid);

**描述**

根据WKB表述和给定的SRID创建一个geometry几何类型对象. 该函数在SQL中充当Geometry Factory角色，如果SRID没有给定默认设置为0，如果输入的的bytea类型参数不是LINESTRING 几何类型对象，返回NULL值，该函数是ST\_LineFromWKB的别名，两者等价

.

**注意**

OGC SPEC 3.2.6.2 - 参数SRID为了满足该规范，和该规范保持一致.

**注意**

如果你十分确定你的几何类型列是LINESTRING,使用ST\_GeomFromWKB性能更高. 而ST\_LineFromWKB函数只是调用ST\_GeomFromWKB这个函数，同时添加了对LINESTRING类型的校验

该方法实现了规范 OpenGIS Simple Features Implementation Speciﬁcation for SQL 1.1. s3.2.6.2

该方法实现了规范 SQL/MM speciﬁcation. SQL-MM 3: 7.2.9

**样例**

SELECT

ST\_LineStringFromWKB(

ST\_AsBinary(ST\_GeomFromText(’LINESTRING(1 2, 3 4)’))

) AS aline,

ST\_LinestringFromWKB(

ST\_AsBinary(ST\_GeomFromText(’POINT(1 2)’))

) IS NULL AS null\_return;

aline

| null\_return

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

010200000002000000000000000000F ... | t

**参考**

ST\_GeomFromWKB, ST\_LineFromWKB









PostGIS 2.2.0dev开发手册

112 / 686

**8.4.22**

**ST\_MakeBox2D**

ST\_MakeBox2D — 根据给定的point几何类型创建BOX2D对象

**用法**

box2d ST\_MakeBox2D(geometry pointLowLeft, geometry pointUpRight);

**描述**

根据给定的point几何类型创建BOX2D对象，该函数对范围查询很有用

**样例**

--Return all features that fall reside or partly reside in a US national atlas coordinate

bounding box

--It is assumed here that the geometries are stored with SRID = 2163 (US National atlas

equal area)

SELECT feature\_id, feature\_name, the\_geom

FROM features

WHERE the\_geom && ST\_SetSRID(ST\_MakeBox2D(ST\_Point(-989502.1875, 528439.5625),

ST\_Point(-987121.375 ,529933.1875)),2163)

**参考**

ST\_MakePoint, ST\_Point, ST\_SetSRID, ST\_SRID

←

←

**8.4.23**

**ST\_3DMakeBox**

ST\_3DMakeBox —根据3D 类型的point几何类型创建BOX3D对象

**用法**

box3d ST\_3DMakeBox(geometry point3DLowLeftBottom, geometry point3DUpRightTop);

**描述**

根据3D 类型的point几何类型创建BOX3D对象.

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

2.0.0版本更新：在先前的版本，该函数曾经的名字是ST\_MakeBox3D

**样例**

SELECT ST\_3DMakeBox(ST\_MakePoint(-989502.1875, 528439.5625, 10),

ST\_MakePoint(-987121.375 ,529933.1875, 10)) As abb3d

--bb3d--

--------

BOX3D(-989502.1875 528439.5625 10,-987121.375 529933.1875 10)



PostGIS 2.2.0dev开发手册

113 / 686

**参考**

ST\_MakePoint, ST\_SetSRID, ST\_SRID

**8.4.24**

**ST\_MakeLine**

ST\_MakeLine — 根据point或line几何类型创建Linestring类型对象

**用法**

geometry ST\_MakeLine(geometry set geoms);

geometry ST\_MakeLine(geometry geom1, geometry geom2);

geometry ST\_MakeLine(geometry[] geoms\_array);

**描述**

ST\_MakeLine函数有3种形式(3种不同的参数)，一种是聚集函数，它用一排point或line几何类型生成一个linestring几何对象。一种是用一个数组的point或line生成一个linestring对象，还有一种函数是用两个point或linestring类型生成一个linestring几何类型对象。你可能需要先把这些点用子查询来对点排序，然后再当做这个函数的参数。当添加了line参数后，输出中将移除共同的点(译者注：就是两个线段可能有重复的点，用两个线段生成一个线段后，输出中重复相同的点会被移除)

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

可用版本: 1.4.0 - ST\_MakeLine(geomarray) 函数被引入，ST\_MakeLine 聚集函数性能提升，处理更多点会更快

可用版本: 2.0.0 –支持 linestring 类型输入的函数被引入

**样例: Spatial Aggregate version**

This example takes a sequence of GPS points and creates one record for each gps travel where the geometry ﬁeld is a line string

composed of the gps points in the order of the travel.

-- For pre-PostgreSQL 9.0 - this usually works,

-- but the planner may on occasion choose not to respect the order of the subquery

SELECT gps.gps\_track, ST\_MakeLine(gps.the\_geom) As newgeom

FROM (SELECT gps\_track,gps\_time, the\_geom

FROM gps\_points ORDER BY gps\_track, gps\_time) As gps

GROUP BY gps.gps\_track;

-- If you are using PostgreSQL 9.0+

-- (you can use the new ORDER BY support for aggregates)

-- this is a guaranteed way to get a correctly ordered linestring

-- Your order by part can order by more than one column if needed

SELECT gps.gps\_track, ST\_MakeLine(gps.the\_geom ORDER BY gps\_time) As newgeom

FROM gps\_points As gps

GROUP BY gps.gps\_track;

**样例: Non-Spatial Aggregate version**

First example is a simple one off line string composed of 2 points. The second formulates line strings from 2 points a user draws.

The third is a one-off that joins 2 3d points to create a line in 3d space.



PostGIS 2.2.0dev开发手册

114 / 686

SELECT ST\_AsText(ST\_MakeLine(ST\_MakePoint(1,2), ST\_MakePoint(3,4)));

st\_astext

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

LINESTRING(1 2,3 4)

SELECT userpoints.id, ST\_MakeLine(startpoint, endpoint) As drawn\_line

FROM userpoints ;

SELECT ST\_AsEWKT(ST\_MakeLine(ST\_MakePoint(1,2,3), ST\_MakePoint(3,4,5)));

st\_asewkt

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

LINESTRING(1 2 3,3 4 5)

**样例: Using Array version**

SELECT ST\_MakeLine(ARRAY(SELECT ST\_Centroid(the\_geom) FROM visit\_locations ORDER BY

visit\_time));

--Making a 3d line with 3 3-d points

SELECT ST\_AsEWKT(ST\_MakeLine(ARRAY[ST\_MakePoint(1,2,3),

ST\_MakePoint(3,4,5), ST\_MakePoint(6,6,6)]));

st\_asewkt

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

LINESTRING(1 2 3,3 4 5,6 6 6)

**参考**

ST\_AsEWKT, ST\_AsText, ST\_GeomFromText, ST\_MakePoint

←

**8.4.25**

**ST\_MakeEnvelope**

ST\_MakeEnvelope — 根据给定的最小值范围和最大值范围生成一个矩形，输入值必须是SRS（spatial\_reference\_system表）规定的SRID值

**用法**

geometry ST\_MakeEnvelope(double precision xmin, double precision ymin, double precision xmax, double precision ymax,

integer srid=unknown);

**描述**

根据给定的最小值范围和最大值范围生成一个矩形，输入值必须是SRS（spatial\_reference\_system表）规定的SRID值，如果SRID只没有给出，SRID会被假定为未知

可用版本: 1.5

版本提升: 2.0: 不指定SRID值也能生成一个envelope类型的函数被引入进来

**样例: Building a bounding box polygon**

SELECT ST\_AsText(ST\_MakeEnvelope(10, 10, 11, 11, 4326));

st\_asewkt

-----------

POLYGON((10 10, 10 11, 11 11, 11 10, 10 10))

PostGIS 2.2.0dev开发手册

115 / 686

**参考**

ST\_MakePoint, ST\_MakeLine, ST\_MakePolygon

**8.4.26**

**ST\_MakePolygon**

ST\_MakePolygon —根据给定的闭合的LineString类型（译者注：闭合的linestring类型请参考https://msdn.microsoft.com/zh-cn/library/bb895372(v=sql.120).aspx）生成一个多边形，输入的几何类型必须是封闭的曲线

**用法**

geometry ST\_MakePolygon(geometry linestring);

geometry ST\_MakePolygon(geometry outerlinestring, geometry[] interiorlinestrings);

**描述**

根据给定的闭合的linestring生成一个多边形，输入的几何类型必须是闭合的linestring.该函数有两种形式：

形式 1: 输入参数是闭合的linestring

形式 2: 根据给定的闭合线段(译者注：看起来像一个外壳、框一样)和洞状的数组(译者注：就是输入一个闭合的linestring里面包含了子linestring)你可以使用ST\_Accumor函数创建一个geometry数组，或根据PostgreSQL ARRAY()函数创建创建ARRAY[]数组，输入的geometry类型必须是闭合的线段

**注意**

这个函数不接受MULTILINESTRING参数. 使用ST\_LineMerge或ST\_Dump生成linestring类型

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

**样例: Single closed LINESTRING**

--2d line

SELECT ST\_MakePolygon(ST\_GeomFromText(’LINESTRING(75.15 29.53,77 29,77.6 29.5, 75.15 29.53) ←

’));

--If linestring is not closed

--you can add the start point to close it

SELECT ST\_MakePolygon(ST\_AddPoint(foo.open\_line, ST\_StartPoint(foo.open\_line)))

FROM (

SELECT ST\_GeomFromText(’LINESTRING(75.15 29.53,77 29,77.6 29.5)’) As open\_line) As foo;

--3d closed line

SELECT ST\_MakePolygon(ST\_GeomFromText(’LINESTRING(75.15 29.53 1,77 29 1,77.6 29.5 1, 75.15

←

29.53 1)’));

st\_asewkt

-----------

POLYGON((75.15 29.53 1,77 29 1,77.6 29.5 1,75.15 29.53 1))

--measured line --

SELECT ST\_MakePolygon(ST\_GeomFromText(’LINESTRINGM(75.15 29.53 1,77 29 1,77.6 29.5 2, 75.15 ←

29.53 2)’));

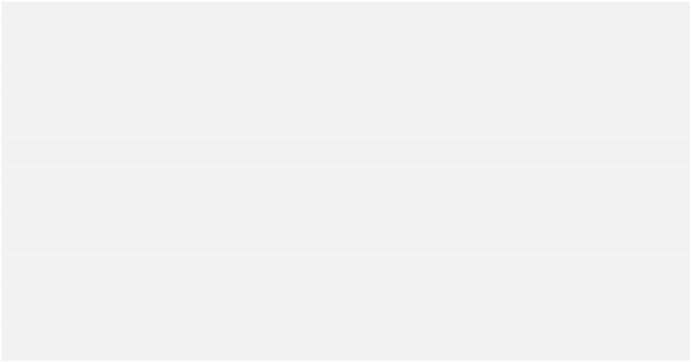
st\_asewkt

----------

POLYGONM((75.15 29.53 1,77 29 1,77.6 29.5 2,75.15 29.53 2))







PostGIS 2.2.0dev开发手册

116 / 686

**样例: Outter shell with inner shells**

Build a donut with an ant hole

SELECT ST\_MakePolygon(

ST\_ExteriorRing(ST\_Buffer(foo.line,10)),

ARRAY[ST\_Translate(foo.line,1,1),

ST\_ExteriorRing(ST\_Buffer(ST\_MakePoint(20,20),1)) ]

)

FROM

(SELECT ST\_ExteriorRing(ST\_Buffer(ST\_MakePoint(10,10),10,10))

As line )

As foo;

根据省份多边形或多边形创建一个省份的边界，这里面的孔表示这个省份里面的一些湖泊,linestring 在这里是用PostGIS ST\_Accum使用样例

**注意**

下面用例的CASE语句使用是因为如果ST\_MakePolygon函数的输入是一个空数组，这个函数的输出也是NULL

**注意**

使用left join方式是为了保证我们要在查询后返回所有省份，即便这个省份没有lakes

SELECT p.gid, p.province\_name,

CASE WHEN

ST\_Accum(w.the\_geom) IS NULL THEN p.the\_geom

ELSE

ST\_MakePolygon(ST\_LineMerge(ST\_Boundary(p.the\_geom)), ST\_Accum(w.the\_geom)) END

FROM

provinces p LEFT JOIN waterlines w

ON (ST\_Within(w.the\_geom, p.the\_geom) AND ST\_IsClosed(w.the\_geom))

GROUP BY p.gid, p.province\_name, p.the\_geom;

--Same example above but utilizing a correlated subquery

--and PostgreSQL built-in ARRAY() function that converts a row set to an array

SELECT p.gid,

p.province\_name, CASE WHEN

EXISTS(SELECT w.the\_geom

FROM waterlines w

WHERE ST\_Within(w.the\_geom, p.the\_geom)

AND ST\_IsClosed(w.the\_geom))

THEN

ST\_MakePolygon(ST\_LineMerge(ST\_Boundary(p.the\_geom)),

ARRAY(SELECT w.the\_geom

FROM waterlines w

WHERE ST\_Within(w.the\_geom, p.the\_geom)

AND ST\_IsClosed(w.the\_geom)))

ELSE p.the\_geom END As the\_geom

FROM

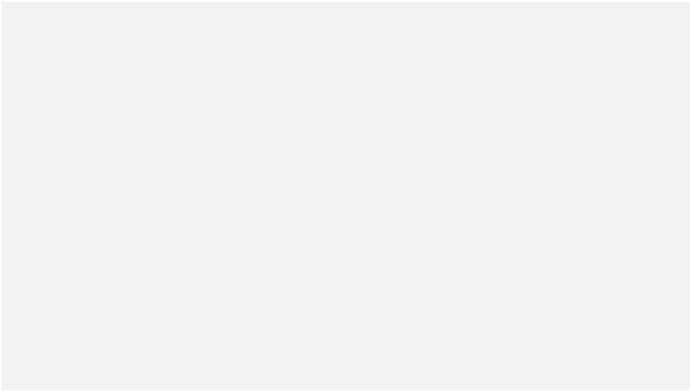
provinces p;

**参考**

ST\_Accum, ST\_AddPoint, ST\_GeometryType, ST\_IsClosed, ST\_LineMerge, ST\_BuildArea







PostGIS 2.2.0dev开发手册

117 / 686

**8.4.27**

**ST\_MakePoint**

ST\_MakePoint — 创建一个2D,3DZ or 4D point 几何类型.

**用法**

geometry ST\_MakePoint(double precision x, double precision y);

geometry ST\_MakePoint(double precision x, double precision y, double precision z);

geometry ST\_MakePoint(double precision x, double precision y, double precision z, double precision m);

**描述**

创建一个2D,3DZ or 4D point 几何类型. ST\_MakePoint函数虽然和OGC不保持一致，但比ST\_GeomFromText 和 ST\_PointFromText通常更快，也更精确。如果你的输入参数是裸坐标而不是WKT描述的对象，使用这个函数也更简单

**注意**

注意x是经度，而y是维度

**注意**

如果你需要使用x,y,m来生成点，可以使用ST\_MakePointM函数

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

**样例**

--Return point with unknown SRID

SELECT ST\_MakePoint(-71.1043443253471, 42.3150676015829);

--Return point marked as WGS 84 long lat

SELECT ST\_SetSRID(ST\_MakePoint(-71.1043443253471, 42.3150676015829),4326);

--Return a 3D point (e.g. has altitude)

SELECT ST\_MakePoint(1, 2,1.5);

--Get z of point

SELECT ST\_Z(ST\_MakePoint(1, 2,1.5));

result

-------

1.5

**参考**

ST\_GeomFromText, ST\_PointFromText, ST\_SetSRID, ST\_MakePointM

**8.4.28**

**ST\_MakePointM**

ST\_MakePointM — 使用x,y,m坐标创建一个point 几何类型对象.







PostGIS 2.2.0dev开发手册

118 / 686

**用法**

geometry ST\_MakePointM(ﬂoat x, ﬂoat y, ﬂoat m);

**描述**

使用x,y,m坐标创建一个point 几何类型对象.

**注意**

x是经度，y是维度.

**样例**

我们在下面的样例中使用ST\_AsEWKT 函数来显示几何对象的文本表述而不是ST\_AsText 函数，因为 ST\_AsText 不支持返回M值

--Return EWKT representation of point with unknown SRID

SELECT ST\_AsEWKT(ST\_MakePointM(-71.1043443253471, 42.3150676015829, 10));

--result

st\_asewkt

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

POINTM(-71.1043443253471 42.3150676015829 10)

--Return EWKT representation of point with measure marked as WGS 84 long lat

SELECT ST\_AsEWKT(ST\_SetSRID(ST\_MakePointM(-71.1043443253471, 42.3150676015829,10),4326));

st\_asewkt

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

SRID=4326;POINTM(-71.1043443253471 42.3150676015829 10)

--Return a 3d point (e.g. has altitude)

SELECT ST\_MakePoint(1, 2,1.5);

--Get m of point

SELECT ST\_M(ST\_MakePointM(-71.1043443253471, 42.3150676015829,10));

result

-------

10

**参考**

ST\_AsEWKT, ST\_MakePoint, ST\_SetSRID

**8.4.29**

**ST\_MLineFromText**

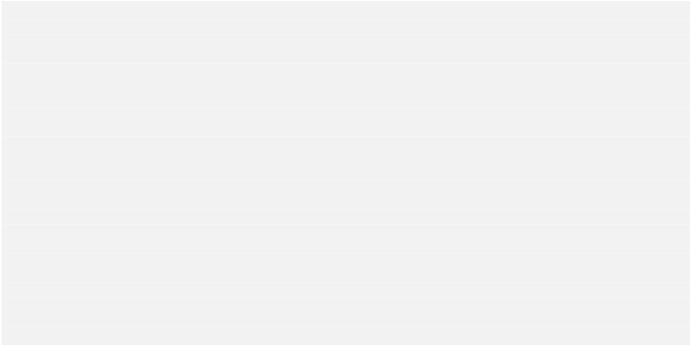
ST\_MLineFromText — 根据WKT表述的几何对象返回ST\_MultiLineString值

**用法**

geometry ST\_MLineFromText(text WKT, integer srid);

geometry ST\_MLineFromText(text WKT);





PostGIS 2.2.0dev开发手册

119 / 686

**描述**

根据WKT表述和给定的SRID创建一个geometry几何类型对象，如果SRID没有给定默认设置为0

OGC SPEC 3.2.6.2 - 参数SRID为了满足该规范，和该规范保持一致

如果输入的WKT对象不是MULTILINESTRING几何类型对象，返回NULL值

**注意**

如果你十分确定你的几何类型列是Point类型,不要使用该函数，它比ST\_GeomFromText慢很多，因为这个函数还增加了对输入类型校验功能

该方法实现了规范 OpenGIS Simple Features Implementation Speciﬁcation for SQL 1.1. s3.2.6.2

该方法实现了规范 SQL/MM speciﬁcation.SQL-MM 3: 9.4.4

**样例**

SELECT ST\_MLineFromText(’MULTILINESTRING((1 2, 3 4), (4 5, 6 7))’);

**参考**

ST\_GeomFromText

**8.4.30**

**ST\_MPointFromText**

ST\_MPointFromText —根据WKT表述和给定的SRID创建一个geometry几何类型对象，如果SRID没有给定默认设置为0

**用法**

geometry ST\_MPointFromText(text WKT, integer srid);

geometry ST\_MPointFromText(text WKT);

**描述**

根据WKT表述和给定的SRID创建一个geometry几何类型对象，如果SRID没有给定默认设置为0.

OGC SPEC 3.2.6.2 - 参数SRID为了满足该规范，和该规范保持一致

如果输入的WKT参数不是MULTIPOINT，返回NULL值

**注意**

.如果你十分确定你的几何类型列是Point类型，不要使用这个函数,使用ST\_GeomFromText性能更高，因为该函数添加了对输入类型的校验步骤

该方法实现了规范 OpenGIS Simple Features Implementation Speciﬁcation for SQL 1.1. 3.2.6.2

该方法实现了规范 SQL/MM speciﬁcation. SQL-MM 3: 9.2.4













PostGIS 2.2.0dev开发手册

120 / 686

**样例**

SELECT ST\_MPointFromText(’MULTIPOINT(1 2, 3 4)’);

SELECT ST\_MPointFromText(’MULTIPOINT(-70.9590 42.1180, -70.9611 42.1223)’, 4326);

**参考**

ST\_GeomFromText

**8.4.31**

**ST\_MPolyFromText**

根据WKT表述和给定的SRID创建一个MultiPolygon Geometry几何类型对象，如果SRID没有给定默认设置为0

**用法**

geometry ST\_MPolyFromText(text WKT, integer srid);

geometry ST\_MPolyFromText(text WKT);

**描述**

根据WKT表述和给定的SRID创建一个MultiPolygon Geometry几何类型对象，如果SRID没有给定默认设置为0

OGC SPEC 3.2.6.2 - 参数SRID为了满足该规范，和该规范保持一致

如果WKT参数不是MULTIPOLYGON，返回null值

**注意**

如果你十分确定你的几何类型列是multipolygon类型，不要使用这个函数,使用ST\_GeomFromText性能更高，因为该函数添加了对输入类型的校验步骤

该方法实现了规范 OpenGIS Simple Features Implementation Speciﬁcation for SQL 1.1. s3.2.6.2

该方法实现了规范 SQL/MM speciﬁcation. SQL-MM 3: 9.6.4

**样例**

SELECT ST\_MPolyFromText(’MULTIPOLYGON(((0 0 1,20 0 1,20 20 1,0 20 1,0 0 1),(5 5 3,5 7 3,7 7 ←

3,7 5 3,5 5 3)))’);

SELECt ST\_MPolyFromText(’MULTIPOLYGON(((-70.916 42.1002,-70.9468 42.0946,-70.9765

←

42.0872,-70.9754 42.0875,-70.9749 42.0879,-70.9752 42.0881,-70.9754 42.0891,-70.9758

42.0894,-70.9759 42.0897,-70.9759 42.0899,-70.9754 42.0902,-70.9756 42.0906,-70.9753

42.0907,-70.9753 42.0917,-70.9757 42.0924,-70.9755 42.0928,-70.9755 42.0942,-70.9751

42.0948,-70.9755 42.0953,-70.9751 42.0958,-70.9751 42.0962,-70.9759 42.0983,-70.9767

42.0987,-70.9768 42.0991,-70.9771 42.0997,-70.9771 42.1003,-70.9768 42.1005,-70.977

42.1011,-70.9766 42.1019,-70.9768 42.1026,-70.9769 42.1033,-70.9775 42.1042,-70.9773

42.1043,-70.9776 42.1043,-70.9778 42.1048,-70.9773 42.1058,-70.9774 42.1061,-70.9779

42.1065,-70.9782 42.1078,-70.9788 42.1085,-70.9798 42.1087,-70.9806 42.109,-70.9807

42.1093,-70.9806 42.1099,-70.9809 42.1109,-70.9808 42.1112,-70.9798 42.1116,-70.9792

42.1127,-70.979 42.1129,-70.9787 42.1134,-70.979 42.1139,-70.9791 42.1141,-70.9987

42.1116,-71.0022 42.1273,

-70.9408 42.1513,-70.9315 42.1165,-70.916 42.1002)))’,4326);

←

←

←

←

←

←

←

←

←

←







PostGIS 2.2.0dev开发手册

121 / 686

**参考**

ST\_GeomFromText, ST\_SRID

**8.4.32**

**ST\_Point**

ST\_Point — 根据给定的坐标值，返回ST\_Point值对应的几何类型对象，这个函数是OGC 函数ST\_MakePoint 的别名

**用法**

geometry ST\_Point(ﬂoat x\_lon, ﬂoat y\_lat);

**描述**

根据给定的坐标值，返回ST\_Point值对应的几何类型对象. 这个函数和SQL/MM规范的ST\_MakePoint函数兼容，ST\_MakePoint这个函数只有x，y两个参数

该方法实现了规范 SQL/MM speciﬁcation. SQL-MM 3: 6.1.2

**样例: Geometry**

SELECT ST\_SetSRID(ST\_Point(-71.1043443253471, 42.3150676015829),4326)

**样例: Geography**

SELECT CAST(ST\_SetSRID(ST\_Point(-71.1043443253471, 42.3150676015829),4326) As geography);

-- the :: is PostgreSQL short-hand for casting.

SELECT ST\_SetSRID(ST\_Point(-71.1043443253471, 42.3150676015829),4326)::geography;

--If your point coordinates are in a different spatial reference from WGS-84 long lat, then ←

you need to transform before casting

-- This example we convert a point in Pennsylvania State Plane feet to WGS 84 and then

geography

SELECT ST\_Transform(ST\_SetSRID(ST\_Point(3637510, 3014852),2273),4326)::geography;

**参考**

Section 4.2.1, ST\_MakePoint, ST\_SetSRID, ST\_Transform

←

**8.4.33**

**ST\_PointFromGeoHash**

ST\_PointFromGeoHash — 根据GeoHash字符串返回一个point几何类型对象

**用法**

point ST\_PointFromGeoHash(text geohash, integer precision=full\_precision\_of\_geohash);



PostGIS 2.2.0dev开发手册

122 / 686

**描述**

根据GeoHash字符串返回一个point几何类型对象. 返回的point是GeoHash的中心点

如果precision参数没有给出，ST\_PointFromGeoHash函数根据输入的GeoHash字符串的全精度返回一个point对象。如果指定了precision参数，ST\_PointFromGeoHash函数会从GeoHash字符串中使用很多字符来创建一个point对象。

可用版本: 2.1.0

**样例**

SELECT ST\_AsText(ST\_PointFromGeoHash(’9qqj7nmxncgyy4d0dbxqz0’));

st\_astext

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

POINT(-115.172816 36.114646)

SELECT ST\_AsText(ST\_PointFromGeoHash(’9qqj7nmxncgyy4d0dbxqz0’, 4));

st\_astext

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

POINT(-115.13671875 36.123046875)

SELECT ST\_AsText(ST\_PointFromGeoHash(’9qqj7nmxncgyy4d0dbxqz0’, 10));

st\_astext

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

POINT(-115.172815918922 36.1146435141563)

**参考**

ST\_GeoHash, ST\_Box2dFromGeoHash, ST\_GeomFromGeoHash

**8.4.34**

**ST\_PointFromText**

ST\_PointFromText —根据WKT表述和给定的SRID创建一个geometry几何类型对象，如果SRID没有给定默认设置为0,即未知

**用法**

geometry ST\_PointFromText(text WKT);

geometry ST\_PointFromText(text WKT, integer srid);

**描述**

根据WKT表述和给定的SRID创建一个geometry几何类型对象，如果SRID没有给定默认设置为0,即未知，如果输入的WKT对象不是point类型，返回为null，如果是无效的WKT描述对象，返回一个错误

**注意**

ST\_PointFromText函数有两种变体(使用方式), 第一种没有SRID参数，返回的geometry类型没有定义空间参考系，第二种定义了第二个参数SRID，返回的geometry对象包含了SRID值作为元数据信息，SRID值必须是定义在表spatial\_ref\_sys中的一种



PostGIS 2.2.0dev开发手册

123 / 686

**注意**

如果你十分确定你的几何类型列是Point类型，不要使用这个函数,使用ST\_GeomFromText性能更高，因为该函数添加了对输入类型的校验步骤.如果你根据经纬度来构造point对象，并且和OGC兼容的函数使用后相比，更关注性能和精度，请使用函数ST\_MakePoint或与OGC的ST\_Point函数

该方法实现了规范 OpenGIS Simple Features Implementation Speciﬁcation for SQL 1.1. s3.2.6.2 -参数SRID为了满足该规范一致性要求

该方法实现了规范 SQL/MM speciﬁcation. SQL-MM 3: 6.1.8

**样例**

SELECT ST\_PointFromText(’POINT(-71.064544 42.28787)’);

SELECT ST\_PointFromText(’POINT(-71.064544 42.28787)’, 4326);

**参考**

ST\_GeomFromText, ST\_MakePoint, ST\_Point, ST\_SRID

**8.4.35**

**ST\_PointFromWKB**

ST\_PointFromWKB —根据WKB表述和给定的SRID创建一个geometry几何类型对象，如果SRID没有给定默认设置为0,即未知

**用法**

geometry ST\_GeomFromWKB(bytea geom);

geometry ST\_GeomFromWKB(bytea geom, integer srid);

**描述**

根据WKB表述和给定的SRID创建一个geometry几何类型对象，如果SRID没有给定默认设置为0,即未知，该函数在SQL中充当Geometry Factory角色，如果SRID没有给定默认设置为0 ，如果输入的的bytea类型参数不是POINT几何类型对象，返回NULL值

该方法实现了规范 OpenGIS Simple Features Implementation Speciﬁcation for SQL 1.1. s3.2.7.2

该方法实现了规范 SQL/MM speciﬁcation. SQL-MM 3: 6.1.9

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

该函数支持CircularString和Curve几何类型对象















PostGIS 2.2.0dev开发手册

124 / 686

**样例**

SELECT

ST\_AsText(

ST\_PointFromWKB(

ST\_AsEWKB(’POINT(2 5)’::geometry)

)

);

st\_astext

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

POINT(2 5)

(1 row)

SELECT

ST\_AsText(

ST\_PointFromWKB(

ST\_AsEWKB(’LINESTRING(2 5, 2 6)’::geometry)

)

);

st\_astext

-----------

(1 row)

**参考**

ST\_GeomFromWKB, ST\_LineFromWKB

**8.4.36**

**ST\_Polygon**

ST\_Polygon — 根据具体的linestring类型对象和SRID创建一个polygon对象

**用法**

geometry ST\_Polygon(geometry aLineString, integer srid);

**描述**

根据具体的linestring类型对象和SRID创建一个polygon对象

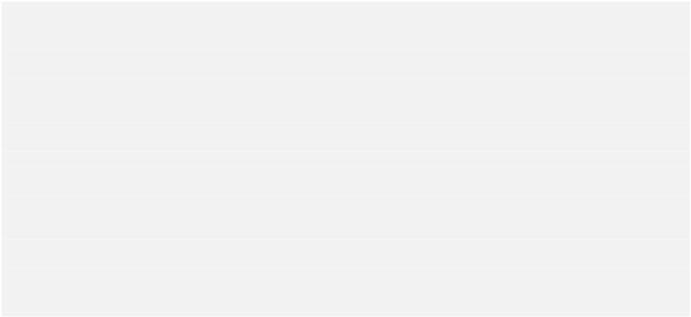
**注意**

ST\_Polygon 函数和该函数的第一个版本的ST\_MakePolygon很像，除了它多了一个设置polygon对象的SRID参数.该函数无法处理MULTILINESTRING类型对象，如果要使用，使用LineMerge函数合并multiline对象.另外不要创建带有孔的polygon对象，如果需要创建带孔的，使用函数ST\_MakePolygon处理

该方法实现了规范 [OpenGIS Simple Features Implementation Speciﬁcation for SQL 1.1.](http://www.opengeospatial.org/standards/sfs)

该方法实现了规范 SQL/MM speciﬁcation. SQL-MM 3: 8.3.2

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.











PostGIS 2.2.0dev开发手册

125 / 686

**样例**

--a 2d polygon

SELECT ST\_Polygon(ST\_GeomFromText(’LINESTRING(75.15 29.53,77 29,77.6 29.5, 75.15 29.53)’),

4326);

--result--

POLYGON((75.15 29.53,77 29,77.6 29.5,75.15 29.53))

--a 3d polygon

SELECT ST\_AsEWKT(ST\_Polygon(ST\_GeomFromEWKT(’LINESTRING(75.15 29.53 1,77 29 1,77.6 29.5 1,

75.15 29.53 1)’), 4326));

result

------

SRID=4326;POLYGON((75.15 29.53 1,77 29 1,77.6 29.5 1,75.15 29.53 1))

**参考**

ST\_AsEWKT, ST\_AsText, ST\_GeomFromEWKT, ST\_GeomFromText, ST\_LineMerge, ST\_MakePolygon

←

←

**8.4.37**

**ST\_PolygonFromText**

ST\_PolygonFromText —根据WKT表述和给定的SRID创建一个geometry几何类型对象，如果SRID没有给定默认设置为0,即未知

**用法**

geometry ST\_PolygonFromText(text WKT);

geometry ST\_PolygonFromText(text WKT, integer srid);

**描述**

根据WKT表述和给定的SRID创建一个geometry几何类型对象，如果SRID没有给定默认设置为0,即未知.如果输入不是WKT表述的polygon对象，返回null值

OGC SPEC 3.2.6.2 - 参数SRID为了满足该规范，和该规范保持一致

**注意**

如果你十分确定你的几何类型列是polygon类型，不要使用这个函数,使用ST\_GeomFromText性能更高，因为该函数添加了对输入类型的校验步骤

该方法实现了规范 OpenGIS Simple Features Implementation Speciﬁcation for SQL 1.1. s3.2.6.2

该方法实现了规范 SQL/MM speciﬁcation. SQL-MM 3: 8.3.6

**样例**

SELECT ST\_PolygonFromText(’POLYGON((-71.1776585052917 42.3902909739571,-71.1776820268866

42.3903701743239,

-71.1776063012595 42.3903825660754,-71.1775826583081 42.3903033653531,-71.1776585052917

42.3902909739571))’);

st\_polygonfromtext

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

←

←







PostGIS 2.2.0dev开发手册

126 / 686

010300000001000000050000006...

SELECT ST\_PolygonFromText(’POINT(1 2)’) IS NULL as point\_is\_notpoly;

point\_is\_not\_poly

----------

t

**参考**

ST\_GeomFromText

**8.4.38**

**ST\_WKBToSQL**

ST\_WKBToSQL —根据WKB描述的对象参数，返回一个geometry类型对象，这个函数是ST\_GeomFromWKB的别名，但ST\_WKBToSQL这个函数没有SRID参数

**用法**

geometry ST\_WKBToSQL(bytea WKB);

**描述**

该方法实现了规范 SQL/MM speciﬁcation. SQL-MM 3: 5.1.36

**参考**

ST\_GeomFromWKB

**8.4.39**

**ST\_WKTToSQL**

ST\_WKTToSQL —根据WKT描述的对象参数，返回一个geometry类型对象，这个函数是ST\_GeomFromWKT的别名

**用法**

geometry ST\_WKTToSQL(text WKT);

**描述**

该方法实现了规范 SQL/MM speciﬁcation. SQL-MM 3: 5.1.34

**参考**

ST\_GeomFromText





PostGIS 2.2.0dev开发手册

127 / 686

**8.5**

**8.5.1**

**几何对象存取函数**

**GeometryType**

GeometryType — 该函数以字符串方式返回几何对象的几何类型，例如’LINESTRING’, ’POLYGON’, ’MULTIPOINT’等等

**用法**

text GeometryType(geometry geomA);

**描述**

该函数以字符串方式返回几何对象的几何类型，例如’LINESTRING’, ’POLYGON’, ’MULTIPOINT’等等

OGC SPEC s2.1.1.1 –返回geometry类型的实例子类型名称，该类型是geometry的子类型成员。实例的类型名称是一个字符串（译者注：LINESTRING’, ’POLYGON’, ’MULTIPOINT’这些类型都是geometry的子类型，都是从这种类型派生出来的，和Java面向对象理解中的继承差不多。）

**注意**

如果输入的geometry几何对象是可度量的类型(即x,y,m格式)，该函数会返回类型字符串为’POINTM’.

提升: 提升: 2.0.0版本支持Polyhedral Surface类型、Triangle类型、TIN类型.

该方法实现了规范 [OpenGIS Simple Features Implementation Speciﬁcation for SQL 1.1.](http://www.opengeospatial.org/standards/sfs)

该函数支持CircularString和Curve几何类型对象

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

该函数支持 Polyhedral Surface类型几何对象.

该函数支持 Triangles 和 Triangulated Irregular Network Surfaces (TIN)类型.

**样例**

SELECT GeometryType(ST\_GeomFromText(’LINESTRING(77.29 29.07,77.42 29.26,77.27 29.31,77.29

29.07)’));

geometrytype

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

LINESTRING

SELECT ST\_GeometryType(ST\_GeomFromEWKT(’POLYHEDRALSURFACE( ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0

0 0)),

((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),

((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),

((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)) )’));

--result

POLYHEDRALSURFACE

←

←













PostGIS 2.2.0dev开发手册

128 / 686

SELECT GeometryType(geom) as result

FROM

(SELECT

ST\_GeomFromEWKT(’TIN (((

0 0 0,

0 0 1,

0 1 0,

0 0 0

)), ((

0 0 0,

0 1 0,

1 1 0,

0 0 0

))

)’)

) AS g;

result

--------

TIN

**参考**

ST\_GeometryType

AS geom

**8.5.2**

**ST\_Boundary**

ST\_Boundary — 返回这个几何类型对象的边界组成的闭包

**用法**

geometry ST\_Boundary(geometry geomA);

**描述**

返回这个几何类型对象的边界组成的闭包. 组合边界的定义与描述参考OGC SPEC规范的章节： 3.12.3.2. 由于该函数的返回值是一个闭包，因此拓扑结构是封闭的，同时结果可以用OGC SPEC规范 3.12.2节讨论的基本的几何类型对象来表示。该函数由GEOS模块提供

**注意**

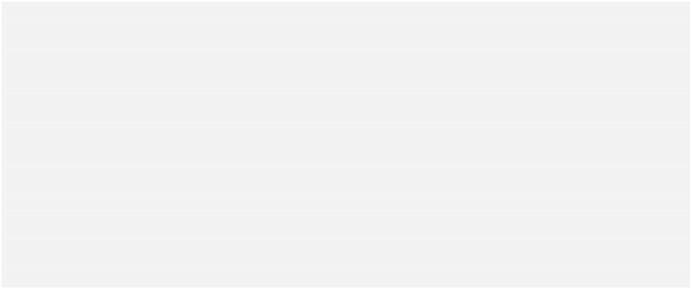
2.0.0之前的版本,如果该函数处理GEOMETRYCOLLECTION对象会抛出异常，从2.0.0版本开始，该函数会返回NULL（对不支持的输入类型）

该方法实现了规范 OpenGIS Simple Features Implementation Speciﬁcation for SQL 1.1. OGC SPEC s2.1.1.1

该方法实现了规范 SQL/MM speciﬁcation. SQL-MM 3: 5.1.14

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

提升: 2.1.0 版本引入对于Triangle类型的支持











PostGIS 2.2.0dev开发手册

129 / 686

**样例**

SELECT ST\_AsText(ST\_Boundary(ST\_GeomFromText(’LINESTRING(1 1,0 0, -1 1)’)));

st\_astext

-----------

MULTIPOINT(1 1,-1 1)

SELECT ST\_AsText(ST\_Boundary(ST\_GeomFromText(’POLYGON((1 1,0 0, -1 1, 1 1))’)));

st\_astext

----------

LINESTRING(1 1,0 0,-1 1,1 1)

--Using a 3d polygon

SELECT ST\_AsEWKT(ST\_Boundary(ST\_GeomFromEWKT(’POLYGON((1 1 1,0 0 1, -1 1 1, 1 1 1))’)));

st\_asewkt

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

LINESTRING(1 1 1,0 0 1,-1 1 1,1 1 1)

--Using a 3d multilinestring

SELECT ST\_AsEWKT(ST\_Boundary(ST\_GeomFromEWKT(’MULTILINESTRING((1 1 1,0 0 0.5, -1 1 1),(1 1

0.5,0 0 0.5, -1 1 0.5, 1 1 0.5) )’)));

st\_asewkt

----------

MULTIPOINT(-1 1 1,1 1 0.75)

**参考**

ST\_ExteriorRing, ST\_MakePolygon

←

**8.5.3**

**ST\_CoordDim**

ST\_CoordDim — 返回ST\_ Geometry 值对应的几何类型的坐标维度值

**用法**

integer ST\_CoordDim(geometry geomA);

**描述**

返回ST\_Geometry 值对应的几何类型对象的坐标维度，该函数等价于SQL/MM规范的ST\_Ndims函数

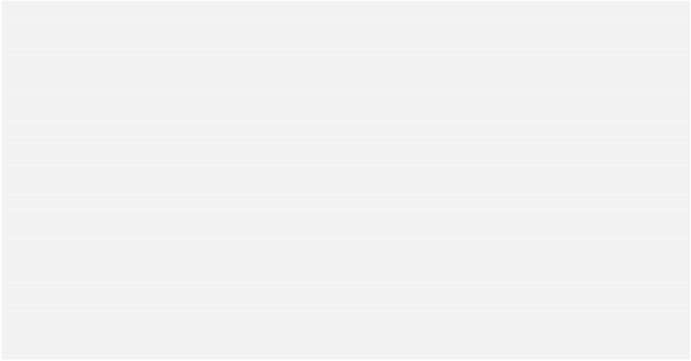
该方法实现了规范 [OpenGIS Simple Features Implementation Speciﬁcation for SQL 1.1.](http://www.opengeospatial.org/standards/sfs)

该方法实现了规范 SQL/MM speciﬁcation. SQL-MM 3: 5.1.3

该函数支持CircularString和Curve几何类型对象

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

改行支持 Polyhedral Surface几何类型对象













PostGIS 2.2.0dev开发手册

130 / 686

该函数支持 Triangles 和 Triangulated Irregular Network Surfaces (TIN)类型.

**样例**

SELECT ST\_CoordDim(’CIRCULARSTRING(1 2 3, 1 3 4, 5 6 7, 8 9 10, 11 12 13)’);

---result--

3

SELECT ST\_CoordDim(ST\_Point(1,2));

--result--

2

**参考**

ST\_NDims

**8.5.4**

**ST\_Dimension**

ST\_Dimension —返回输入的几何对象固有的维度值，这个维度值必须小于或等于坐标系的维度值

**用法**

integer ST\_Dimension(geometry g);

**描述**

返回输入的几何对象固有的维度值，这个维度值必须小于或等于坐标系的维度值.按照 OGC SPEC标准

s2.1.1.1节 - Point的固有维度值是0，LINESTRING固有维度值是1， POLYGON固有维度值为2, GEOMETRYCOLLECTION 的维度值最大，如果是空几何类型，就返回NULL值.

该方法实现了规范 SQL/MM speciﬁcation. SQL-MM 3: 5.1.2

提升: 2.0.0版本支持Polyhedral Surface类型和TIN类型，如果输入参数为空，也不再抛出异常

**注意**

2.0.0之前的版本, 如果输入参数是空几何类型，该函数会抛出异常

该函数支持 Polyhedral Surface类型几何对象.

该函数支持 Triangles 和 Triangulated Irregular Network Surfaces (TIN)类型.

**样例**

SELECT ST\_Dimension(’GEOMETRYCOLLECTION(LINESTRING(1 1,0 0),POINT(0 0))’);

ST\_Dimension

-----------

1











PostGIS 2.2.0dev开发手册

131 / 686

**参考**

ST\_NDims

**8.5.5**

**ST\_EndPoint**

ST\_EndPoint — 返回一个LINESTRING的最后一个point对象

**用法**

boolean ST\_EndPoint(geometry g);

**描述**

返回一个线段的最后一个point对象，如果输入参数不是LINESTRING类型，则返回为NULL值

该方法实现了规范 SQL/MM speciﬁcation. SQL-MM 3: 7.1.4

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

**注意**

该函数2.0.0版本更新：不再对单MULTILINESTRING类型支持。在旧版本的PostGIS中，该函数能够很好的支持单MULTILINESTRING类型，并返回起始点，从2.0.0开始，该函数会返回NULL值。旧的方式是不规范的特性，但对于那些把数据存储为MULTILINESTRING的用户来说，在2.0.0版本的PostGIS将会返回NULL值

**样例**

postgis=# SELECT ST\_AsText(ST\_EndPoint(’LINESTRING(1 1, 2 2, 3 3)’::geometry));

st\_astext

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

POINT(3 3)

(1 row)

postgis=# SELECT ST\_EndPoint(’POINT(1 1)’::geometry) IS NULL AS is\_null;

is\_null

----------

t

(1 row)

--3d endpoint

SELECT ST\_AsEWKT(ST\_EndPoint(’LINESTRING(1 1 2, 1 2 3, 0 0 5)’));

st\_asewkt

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

POINT(0 0 5)

(1 row)

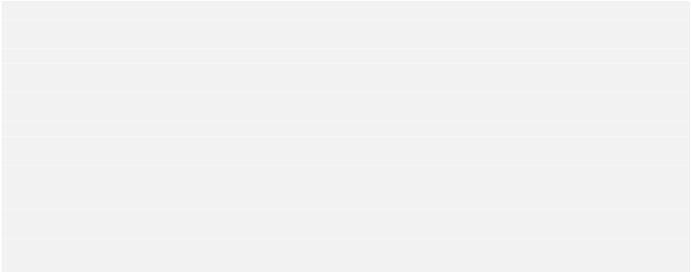
**参考**

ST\_PointN, ST\_StartPoint









PostGIS 2.2.0dev开发手册

132 / 686

**8.5.6**

**ST\_Envelope**

ST\_Envelope —从提供的geometry类型中返回一个box的边界值精度为float8的geometry类型（译者注：是一个凸包）

**用法**

geometry ST\_Envelope(geometry g1);

**描述**

从提供的geometry类型中返回一个box的边界值精度为float8的最小geometry类型（译者注：这是一个闭包）. 多边形是bonding box的各个点((MINX, MINY), (MINX, MAXY), (MAXX, MAXY), (MAXX, MINY), (MINX, MINY))来定义并连接组成的，

. (PostGIS 将会添加一个ZMIN/ZMAX 坐标).

退化的情况例如垂线或者点将会返回一个比POLYGON类型低的维度类型，例如POINT或LINESTRING类型

该函数可用版本: 1.5.0 版本的输出的精度是float8，而不是float4

该方法实现了规范 OpenGIS Simple Features Implementation Speciﬁcation for SQL 1.1. s2.1.1.1

该方法实现了规范 SQL/MM speciﬁcation. SQL-MM 3: 5.1.15

**样例**

SELECT ST\_AsText(ST\_Envelope(’POINT(1 3)’::geometry));

st\_astext

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

POINT(1 3)

(1 row)

SELECT ST\_AsText(ST\_Envelope(’LINESTRING(0 0, 1 3)’::geometry));

st\_astext

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

POLYGON((0 0,0 3,1 3,1 0,0 0))

(1 row)

SELECT ST\_AsText(ST\_Envelope(’POLYGON((0 0, 0 1, 1.0000001 1, 1.0000001 0, 0 0))’::geometry ←

));

st\_astext

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

POLYGON((0 0,0 1,1.00000011920929 1,1.00000011920929 0,0 0))

(1 row)

SELECT ST\_AsText(ST\_Envelope(’POLYGON((0 0, 0 1, 1.0000000001 1, 1.0000000001 0, 0 0))’:: ←

geometry));

st\_astext

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

POLYGON((0 0,0 1,1.00000011920929 1,1.00000011920929 0,0 0))

(1 row)

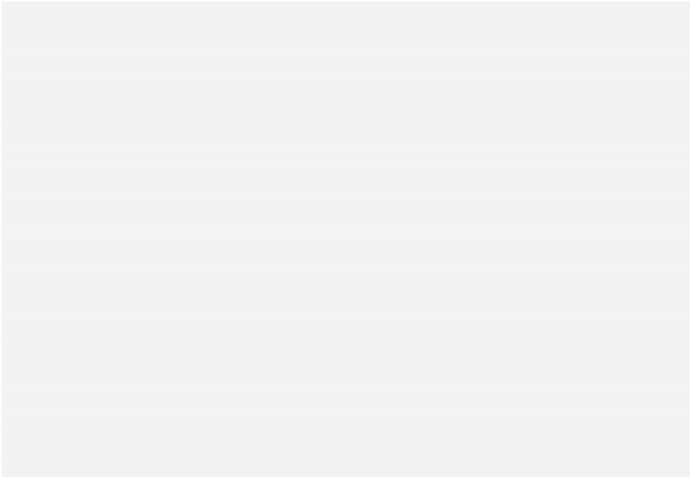
SELECT Box3D(geom), Box2D(geom), ST\_AsText(ST\_Envelope(geom)) As envelopewkt

FROM (SELECT ’POLYGON((0 0, 0 1000012333334.34545678, 1.0000001 1, 1.0000001 0, 0 0))’:: ←

geometry As geom) As foo;







PostGIS 2.2.0dev开发手册

133 / 686

**参考**

Box2D, Box3D

**8.5.7**

**ST\_ExteriorRing**

ST\_ExteriorRing — 返回一个POLYGON 几何类型的外环，如果输入类型不是POLYGON类型，返回NULL值，该函数不支持MULTIPOLYGON

**用法**

geometry ST\_ExteriorRing(geometry a\_polygon);

**描述**

返回一个POLYGON 几何类型的外环，如果输入类型不是POLYGON类型，返回NULL值

**注意**

只支持POLYGON几何类型

该方法实现了规范 OpenGIS Simple Features Implementation Speciﬁcation for SQL 1.1. 2.1.5.1

该方法实现了规范 SQL/MM speciﬁcation. SQL-MM 3: 8.2.3, 8.3.3

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

**样例**

--If you have a table of polygons

SELECT gid, ST\_ExteriorRing(the\_geom) AS ering

FROM sometable;

--If you have a table of MULTIPOLYGONs

--and want to return a MULTILINESTRING composed of the exterior rings of each polygon

SELECT gid, ST\_Collect(ST\_ExteriorRing(the\_geom)) AS erings

FROM (SELECT gid, (ST\_Dump(the\_geom)).geom As the\_geom

FROM sometable) As foo

GROUP BY gid;

--3d Example

SELECT ST\_AsEWKT(

ST\_ExteriorRing(

ST\_GeomFromEWKT(’POLYGON((0 0 1, 1 1 1, 1 2 1, 1 1 1, 0 0 1))’)

)

);

st\_asewkt

---------

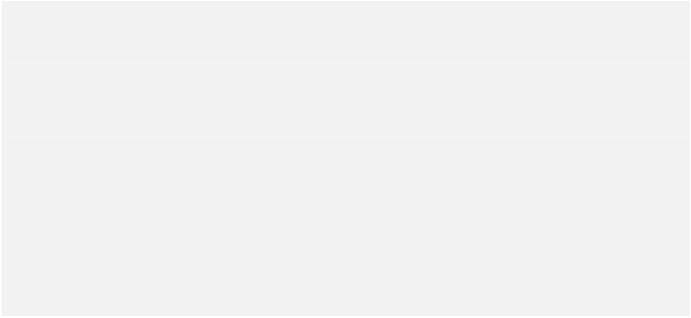
LINESTRING(0 0 1,1 1 1,1 2 1,1 1 1,0 0 1)











PostGIS 2.2.0dev开发手册

134 / 686

**参考**

ST\_InteriorRingN, ST\_Boundary, ST\_NumInteriorRings

**8.5.8**

**ST\_GeometryN**

ST\_GeometryN — 该函数的输入参数是一个collection对象和一个整数N值，返回这个collection里面的第N个对象，这些collection类型可以是GEOMETRYCOLLECTION, (MULTI)POINT, (MULTI)LINES，MULTICURVE or (MULTI)POLYGON, POLYHEDRALSURFACE，其他值返回为NULL

**用法**

geometry ST\_GeometryN(geometry geomA, integer n);

**描述**

该函数的输入参数是一个collection对象和一个整数N值，返回这个collection里面的第N+1个对象，这些collection类型可以是GEOMETRYCOLLECTION, (MULTI)POINT, (MULTI)LINES，MULTICURVE or (MULTI)POLYGON, POLYHEDRALSURFACE，其他值返回为NULL

**注意**

OGC SPEC 0.8.0规范开始，下标是从1开始，之前的版本从0开始

**注意**

如果你想得到所有的几何对象元素，ST\_Dump函数更高效，对单几何类型（译者注：非collection类型的对象

提升: 2.0.0 版本开始支持Polyhedral Surface类型, Triangles and TIN 也开始支持

更新: 2.0.0 ，2.0.0之前的版本对于单几何类型对象会返回NULL值，从2.0.0版本起，对于ST\_GeometryN(..,1) 这样的情况将会返回geometry类型对象

该方法实现了规范 [OpenGIS Simple Features Implementation Speciﬁcation for SQL 1.1.](http://www.opengeospatial.org/standards/sfs)

该方法实现了规范 SQL/MM speciﬁcation. SQL-MM 3: 9.1.5

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

该函数支持CircularString和Curve几何类型对象

该函数支持 Polyhedral Surface类型几何对象.

该函数支持 Triangles 和 Triangulated Irregular Network Surfaces (TIN)类型

译者注：在原文给出的样例之前，可以看下面的样例更好理解一些

CREATE TABLE districts (dist\_id integer, shape sde.st\_geometry);

INSERT INTO districts (dist\_id, shape) VALUES (1,

sde.st\_multipolygon ('multipolygon (((-1 -1, -1 11, 11 11, 11 -1, -1 -1),

(19 -1, 19 11, 29 9, 31 -1, 19 -1), (39 -1, 39 11, 51 11, 51 -1, 39 -1)))', 0)

);

SELECT st\_astext (sde.st\_geometryn (shape, 2)) AS Second\_Element

FROM districts;

Second\_Element

POLYGON ((39 -1, 51 -1, 51 11, 39 11, 39 -1))

.

















PostGIS 2.2.0dev开发手册

135 / 686

**标准 样例**

--Extracting a subset of points from a 3d multipoint

SELECT n, ST\_AsEWKT(ST\_GeometryN(the\_geom, n)) As geomewkt

FROM (

VALUES (ST\_GeomFromEWKT(’MULTIPOINT(1 2 7, 3 4 7, 5 6 7, 8 9 10)’) ),

( ST\_GeomFromEWKT(’MULTICURVE(CIRCULARSTRING(2.5 2.5,4.5 2.5, 3.5 3.5), (10 11, 12 11))’) )

)As foo(the\_geom)

CROSS JOIN generate\_series(1,100) n

WHERE n <= ST\_NumGeometries(the\_geom);

n |

geomewkt

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

1 | POINT(1 2 7)

2 | POINT(3 4 7)

3 | POINT(5 6 7)

4 | POINT(8 9 10)

1 | CIRCULARSTRING(2.5 2.5,4.5 2.5,3.5 3.5)

2 | LINESTRING(10 11,12 11)

--Extracting all geometries (useful when you want to assign an id)

SELECT gid, n, ST\_GeometryN(the\_geom, n)

FROM sometable CROSS JOIN generate\_series(1,100) n

WHERE n <= ST\_NumGeometries(the\_geom);

**Polyhedral Surfaces, TIN and Triangle 样例**

-- Polyhedral surface example

-- Break a Polyhedral surface into its faces

SELECT ST\_AsEWKT(ST\_GeometryN(p\_geom,3)) As geom\_ewkt

FROM (SELECT ST\_GeomFromEWKT(’POLYHEDRALSURFACE(

((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)),

((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)),

((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),

((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),

((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)),

((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1))

)’)

AS p\_geom )

AS a;

geom\_ewkt

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

POLYGON((0 0 0,1 0 0,1 0 1,0 0 1,0 0 0))

-- TIN --

SELECT ST\_AsEWKT(ST\_GeometryN(geom,2)) as wkt

FROM

(SELECT

ST\_GeomFromEWKT(’TIN (((

0 0 0,

0 0 1,

0 1 0,

0 0 0

)), ((

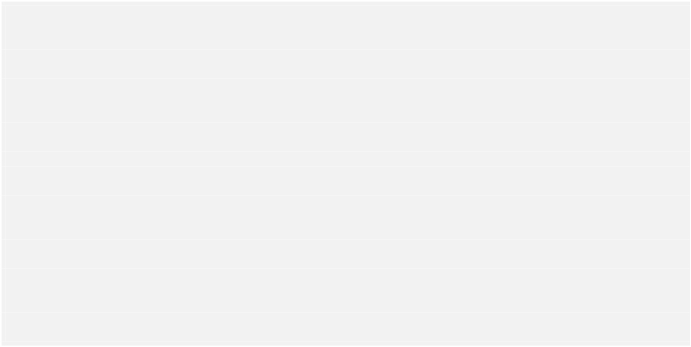
0 0 0,

0 1 0,

1 1 0,

0 0 0

))



PostGIS 2.2.0dev开发手册

136 / 686

)’)

AS geom

) AS g;

-- result --

wkt

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

TRIANGLE((0 0 0,0 1 0,1 1 0,0 0 0))

**参考**

ST\_Dump, ST\_NumGeometries

**8.5.9**

**ST\_GeometryType**

ST\_GeometryType — 返回ST\_Geometry值对应的几何对象的几何类型

**用法**

text ST\_GeometryType(geometry g1);

**描述**

返回的geometry类型是一个字符串，比如’ST\_Linestring’, ’ST\_Polygon’,’ST\_MultiPolygon’ 等待.该函数与函数

GeometryType(geometry) 在string类型结果不同，该函数返回的字符串有前缀ST,同样对geometry类型是否有m值也是一样的处理

提升: 2.0.0版本添加了对Polyhedral Surface类型的引入支持

该方法实现了规范 SQL/MM speciﬁcation. SQL-MM 3: 5.1.4

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

该函数支持 Polyhedral Surface类型几何对象.

**样例**

SELECT ST\_GeometryType(ST\_GeomFromText(’LINESTRING(77.29 29.07,77.42 29.26,77.27

29.31,77.29 29.07)’));

--result

ST\_LineString

←

SELECT ST\_GeometryType(ST\_GeomFromEWKT(’POLYHEDRALSURFACE( ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0

0 0)),

((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),

((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),

((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)) )’));

--result

ST\_PolyhedralSurface

SELECT ST\_GeometryType(ST\_GeomFromEWKT(’POLYHEDRALSURFACE( ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0

0 0)),

((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),

((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),

((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)) )’));

--result

ST\_PolyhedralSurface

←

←







PostGIS 2.2.0dev开发手册

137 / 686

SELECT ST\_GeometryType(geom) as result

FROM

(SELECT

ST\_GeomFromEWKT(’TIN (((

0 0 0,

0 0 1,

0 1 0,

0 0 0

)), ((

0 0 0,

0 1 0,

1 1 0,

0 0 0

))

) AS g;

result

--------

ST\_Tin

**参考**

GeometryType

)’)

AS geom

**8.5.10**

**ST\_InteriorRingN**

ST\_InteriorRingN — 返回一个POLYGON几何类型的第N个内环，如果输入的geometry类型不是POLYGON类型或者N超出了范围，返回NULL值

**用法**

geometry ST\_InteriorRingN(geometry a\_polygon, integer n);

**描述**

返回一个POLYGON几何类型的第N个内环，如果输入的geometry类型不是POLYGON类型或者N超出了范围，返回NULL值。下标从1开始

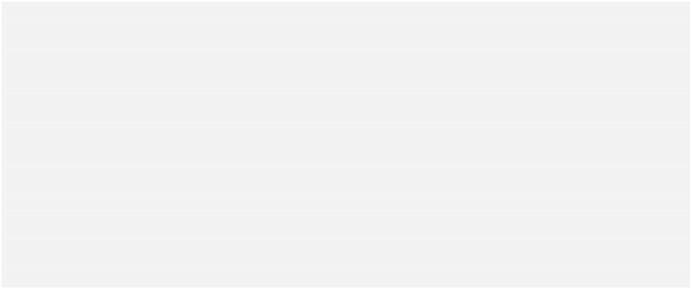
**注意**

该函数不支持MULTIPOLYGON类型. 对于MULTIPOLYGON类型，使用ST\_Dump

该方法实现了规范 [OpenGIS Simple Features Implementation Speciﬁcation for SQL 1.1.](http://www.opengeospatial.org/standards/sfs)

该方法实现了规范 SQL/MM speciﬁcation. SQL-MM 3: 8.2.6, 8.3.5

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.











PostGIS 2.2.0dev开发手册

138 / 686

**样例**

SELECT ST\_AsText(ST\_InteriorRingN(the\_geom, 1)) As the\_geom

FROM (SELECT ST\_BuildArea(

ST\_Collect(ST\_Buffer(ST\_Point(1,2), 20,3),

ST\_Buffer(ST\_Point(1, 2), 10,3))) As the\_geom

)

as foo

**参考**

ST\_ExteriorRing ST\_BuildArea, ST\_Collect, ST\_Dump, ST\_NumInteriorRing, ST\_NumInteriorRings

**8.5.11**

**ST\_IsClosed**

ST\_IsClosed — 如果LINESTRING是闭合的，即起点和终点重合，那么该函数返回值为TRUE，对于Polyhedral surface类型来说，它可能是闭合的

**用法**

boolean ST\_IsClosed(geometry g);

**描述**

如果LINESTRING是闭合的，即起点和终点重合，那么该函数返回值为TRUE，对于Polyhedral surface类型来说,该函数返回值可能是非闭合的面或者闭合的"壳"(译者注：参考下面的样例就很容易理解了)

该方法实现了规范 [OpenGIS Simple Features Implementation Speciﬁcation for SQL 1.1.](http://www.opengeospatial.org/standards/sfs)

该方法实现了规范 SQL/MM speciﬁcation. SQL-MM 3: 7.1.5, 9.3.3

**Note**

SQL-MM deﬁnes the result of ST\_IsClosed(NULL) to be 0, while PostGIS returns NULL.

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

该函数支持CircularString和Curve几何类型对象

Enhanced: 2.0.0 support for Polyhedral surfaces was introduced.

该函数支持 Polyhedral Surface类型几何对象.

**Line String and Point 样例**

postgis=# SELECT ST\_IsClosed(’LINESTRING(0 0, 1 1)’::geometry);

st\_isclosed

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

f

(1 row)













PostGIS 2.2.0dev开发手册

139 / 686

postgis=# SELECT ST\_IsClosed(’LINESTRING(0 0, 0 1, 1 1, 0 0)’::geometry);

st\_isclosed

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

t

(1 row)

postgis=# SELECT ST\_IsClosed(’MULTILINESTRING((0 0, 0 1, 1 1, 0 0),(0 0, 1 1))’::geometry);

st\_isclosed

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

f

(1 row)

postgis=# SELECT ST\_IsClosed(’POINT(0 0)’::geometry);

st\_isclosed

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

t

(1 row)

postgis=# SELECT ST\_IsClosed(’MULTIPOINT((0 0), (1 1))’::geometry);

st\_isclosed

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

t

(1 row)

**Polyhedral Surface 样例**

-- A cube --

SELECT ST\_IsClosed(ST\_GeomFromEWKT(’POLYHEDRALSURFACE( ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0

0 0)),

((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),

((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),

((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)) )’));

st\_isclosed

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

t

-- Same as cube but missing a side --

SELECT ST\_IsClosed(ST\_GeomFromEWKT(’POLYHEDRALSURFACE( ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0

0)),

((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),

((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),

((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)) )’));

st\_isclosed

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

f

**参考**

ST\_IsRing

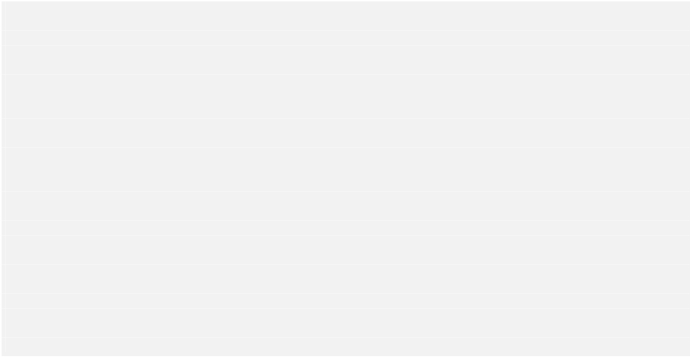
←

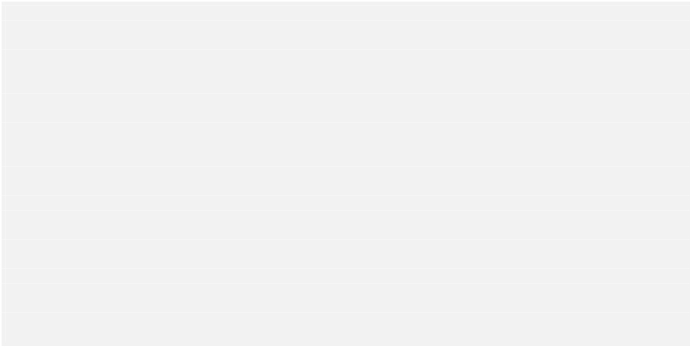
←

**8.5.12**

**ST\_IsCollection**

ST\_IsCollection — 如果参数是一个collection类型的几何对象如 (MULTI\*, GEOMETRYCOLLECTION, ...)，则返回值为TRUE





PostGIS 2.2.0dev开发手册

140 / 686

**用法**

boolean ST\_IsCollection(geometry g);

**描述**

如果参数是一个collection类型的几何对象是下面的之一，则返回值为TRUE:

· GEOMETRYCOLLECTION

· MULTI{POINT,POLYGON,LINESTRING,CURVE,SURFACE}

· COMPOUNDCURVE

**注意**

该函数是通过分析geometry类型来判断的，这意味着如果一个collection是空的或者包含单个元素，它也会返回TRUE

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

该函数支持CircularString和Curve几何类型对象

**样例**

postgis=# SELECT ST\_IsCollection(’LINESTRING(0 0, 1 1)’::geometry);

st\_iscollection

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

f

(1 row)

postgis=# SELECT ST\_IsCollection(’MULTIPOINT EMPTY’::geometry);

st\_iscollection

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

t

(1 row)

postgis=# SELECT ST\_IsCollection(’MULTIPOINT((0 0))’::geometry);

st\_iscollection

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

t

(1 row)

postgis=# SELECT ST\_IsCollection(’MULTIPOINT((0 0), (42 42))’::geometry);

st\_iscollection

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

t

(1 row)

postgis=# SELECT ST\_IsCollection(’GEOMETRYCOLLECTION(POINT(0 0))’::geometry);

st\_iscollection

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

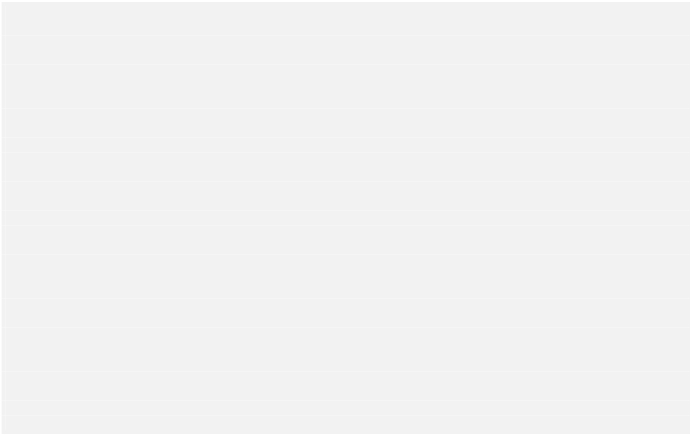
t

(1 row)









PostGIS 2.2.0dev开发手册

141 / 686

**参考**

ST\_NumGeometries

**8.5.13**

**ST\_IsEmpty**

ST\_IsEmpty — 如果输入的geometry参数是空的geometry collection或者空的polygon或者空的point类型等等，那么返回值将会是TRUE

**用法**

boolean ST\_IsEmpty(geometry geomA);

**描述**

如果输入的geometry参数是空的geometry collection或者空的polygon或者空的point类型等等，那么返回值将会是TRUE.

**注意**

SQL-MM 规范定义了ST\_IsEmpty(NULL) 值为 0, 而PostGIS 返回值为 NULL.

该方法实现了规范 OpenGIS Simple Features Implementation Speciﬁcation for SQL 1.1. s2.1.1.1

该方法实现了规范 SQL/MM speciﬁcation. SQL-MM 3: 5.1.7

该函数支持CircularString和Curve几何类型对象

**警告**

更新: 2.0.0 在之前的PostGIS版本，ST\_GeomFromText(’GEOMETRYCOLLECTION(EMPTY)’) 是允许的，在PostGIS 2.0.0版本及之后，为了更好的遵守SQL/MM规范，这种写法是禁止的，不符合语法的

**样例**

SELECT ST\_IsEmpty(ST\_GeomFromText(’GEOMETRYCOLLECTION EMPTY’));

st\_isempty

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

t

(1 row)

SELECT ST\_IsEmpty(ST\_GeomFromText(’POLYGON EMPTY’));

st\_isempty

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

t

(1 row)

SELECT ST\_IsEmpty(ST\_GeomFromText(’POLYGON((1 2, 3 4, 5 6, 1 2))’));

st\_isempty

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











PostGIS 2.2.0dev开发手册

142 / 686

f

(1 row)

SELECT ST\_IsEmpty(ST\_GeomFromText(’POLYGON((1 2, 3 4, 5 6, 1 2))’)) = false;

?column?

----------

t

(1 row)

SELECT ST\_IsEmpty(ST\_GeomFromText(’CIRCULARSTRING EMPTY’));

st\_isempty

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

t

(1 row)

**8.5.14**

**ST\_IsRing**

ST\_IsRing —如果LINESTRING是简单、闭合的，则返回TRUE

**用法**

boolean ST\_IsRing(geometry g);

**描述**

如果LINESTRING是简单、闭合的，则返回TRUE即满足 ST\_IsClosed (ST\_StartPoint((g)) ~= ST\_Endpoint((g))) 并且 ST\_IsSimple (LINESTRING本身不自我相交).

该方法实现了规范 OpenGIS Simple Features Implementation Speciﬁcation for SQL 1.1. 2.1.5.1

该方法实现了规范 SQL/MM speciﬁcation. SQL-MM 3: 7.1.6

**注意**

SQL-MM 规范定义了ST\_ IsRing (NULL) 值为 0, 而PostGIS 返回值为 NULL.

**样例**

SELECT ST\_IsRing(the\_geom), ST\_IsClosed(the\_geom), ST\_IsSimple(the\_geom)

FROM (SELECT ’LINESTRING(0 0, 0 1, 1 1, 1 0, 0 0)’::geometry AS the\_geom) AS foo;

st\_isring | st\_isclosed | st\_issimple

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

t

| t

| t

(1 row)

SELECT ST\_IsRing(the\_geom), ST\_IsClosed(the\_geom), ST\_IsSimple(the\_geom)

FROM (SELECT ’LINESTRING(0 0, 0 1, 1 0, 1 1, 0 0)’::geometry AS the\_geom) AS foo;

st\_isring | st\_isclosed | st\_issimple

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

f

(1 row)

| t

| f







PostGIS 2.2.0dev开发手册

143 / 686

**参考**

ST\_IsClosed, ST\_IsSimple, ST\_StartPoint, ST\_EndPoint

**8.5.15**

**ST\_IsSimple**

ST\_IsSimple — 如果输入的geometry类型没有不规则的几何点，比如自我相交或者自我相切，则返回值为TRUE

**用法**

boolean ST\_IsSimple(geometry geomA);

**描述**

如果输入的geometry类型没有不规则的几何点，比如自我相交或者自我相切，则返回值为TRUE .关于更多的OGC定义的geometry对象的简单性和有效性，参考"Ensuring OpenGIS compliancy of geometries"

**注意**

SQL-MM 规范定义了ST\_ IsSimple (NULL) 值为 0, 而PostGIS 返回值为 NULL.

该方法实现了规范 OpenGIS Simple Features Implementation Speciﬁcation for SQL 1.1. s2.1.1.1

该方法实现了规范 SQL/MM speciﬁcation. SQL-MM 3: 5.1.8

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

**样例**

SELECT ST\_IsSimple(ST\_GeomFromText(’POLYGON((1 2, 3 4, 5 6, 1 2))’));

st\_issimple

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

t

(1 row)

SELECT ST\_IsSimple(ST\_GeomFromText(’LINESTRING(1 1,2 2,2 3.5,1 3,1 2,2 1)’));

st\_issimple

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

f

(1 row)

**参考**

ST\_IsValid

**8.5.16**

**ST\_IsValid**

ST\_IsValid —如果ST\_ Geometry返回的值是满足WKT或EWKT或WKB或EWKB描述的，那么这个函数返回值是TRUE

译者注：返回ST\_Geometry是否是良好的格式，就是比如说一个多边形，那一定是多个点，如果只有一个点，就是有问题的









PostGIS 2.2.0dev开发手册

144 / 686

**用法**

boolean ST\_IsValid(geometry g);

boolean ST\_IsValid(geometry g, integer ﬂags);

**描述**

测试ST\_Geometry 的值是否是符合规范的格式。如果geometry是无效的， PostgreSQL 会提示该对象. 更多关于OGC规范定义的简单性和有效性，请参考 "Ensuring OpenGIS compliancy of geometries"

**注意**

SQL-MM 规范定义了ST\_ IsValid (NULL) 值为 0, 而PostGIS 返回值为 NULL.

从2.0.0版本开始并且GEOS版本在3.3.0及更高版本，才接受参数flags，这样版本将不输出解释geometry无效原因的提示，flags可以使用的值，请查看函数ST\_IsValidDetail说明

该方法实现了规范 [OpenGIS Simple Features Implementation Speciﬁcation for SQL 1.1.](http://www.opengeospatial.org/standards/sfs)

该方法实现了规范 SQL/MM speciﬁcation. SQL-MM 3: 5.1.9

**样例**

SELECT ST\_IsValid(ST\_GeomFromText(’LINESTRING(0 0, 1 1)’)) As good\_line,

ST\_IsValid(ST\_GeomFromText(’POLYGON((0 0, 1 1, 1 2, 1 1, 0 0))’)) As bad\_poly

--results

NOTICE:

Self-intersection at or near point 0 0

good\_line | bad\_poly

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

t

| f

**参考**

ST\_IsSimple, ST\_IsValidReason, ST\_IsValidDetail, ST\_Summary

**8.5.17**

**ST\_IsValidReason**

ST\_IsValidReason —返回一个geometry是否是有效的，以及无效的原因

**用法**

text ST\_IsValidReason(geometry geomA);

text ST\_IsValidReason(geometry geomA, integer ﬂags);

**描述**

返回一个geometry是否是有效的，以及无效的原因.

与函数ST\_IsValid一起使用非常有用，可以用来查找生成无效的几何对象和无效原因的明细报告

flags可以使用的值，请查看函数ST\_IsValidDetail说明

可用版本: 1.4 - 需要 GEOS >= 3.1.0.

可用版本: 2.0 - 需要 GEOS >= 3.3.0以上版本来接收参数flags







PostGIS 2.2.0dev开发手册

145 / 686

**样例**

--First 3 Rejects from a successful quintuplet experiment

SELECT gid, ST\_IsValidReason(the\_geom) as validity\_info

FROM

(SELECT ST\_MakePolygon(ST\_ExteriorRing(e.buff), ST\_Accum(f.line)) As the\_geom, gid

FROM (SELECT ST\_Buffer(ST\_MakePoint(x1\*10,y1), z1) As buff, x1\*10 + y1\*100 + z1\*1000 As gid

FROM generate\_series(-4,6) x1

CROSS JOIN generate\_series(2,5) y1

CROSS JOIN generate\_series(1,8) z1

WHERE x1 > y1\*0.5 AND z1 < x1\*y1) As e

INNER JOIN (SELECT ST\_Translate(ST\_ExteriorRing(ST\_Buffer(ST\_MakePoint(x1\*10,y1), z1)),y1 ←

\*1, z1\*2) As line

FROM generate\_series(-3,6) x1

CROSS JOIN generate\_series(2,5) y1

CROSS JOIN generate\_series(1,10) z1

WHERE x1 > y1\*0.75 AND z1 < x1\*y1) As f

ON (ST\_Area(e.buff) > 78 AND ST\_Contains(e.buff, f.line))

GROUP BY gid, e.buff) As quintuplet\_experiment

WHERE ST\_IsValid(the\_geom) = false

ORDER BY gid

LIMIT 3;

gid

|

validity\_info

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

5330 | Self-intersection [32 5]

5340 | Self-intersection [42 5]

5350 | Self-intersection [52 5]

--simple example

SELECT ST\_IsValidReason(’LINESTRING(220227 150406,2220227 150407,222020 150410)’);

st\_isvalidreason

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

Valid Geometry

**参考**

ST\_IsValid, ST\_Summary

**8.5.18**

**ST\_IsValidDetail**

ST\_IsValidDetail — 返回几何类型是否有效以及无效原因和无效的几何对象是哪些

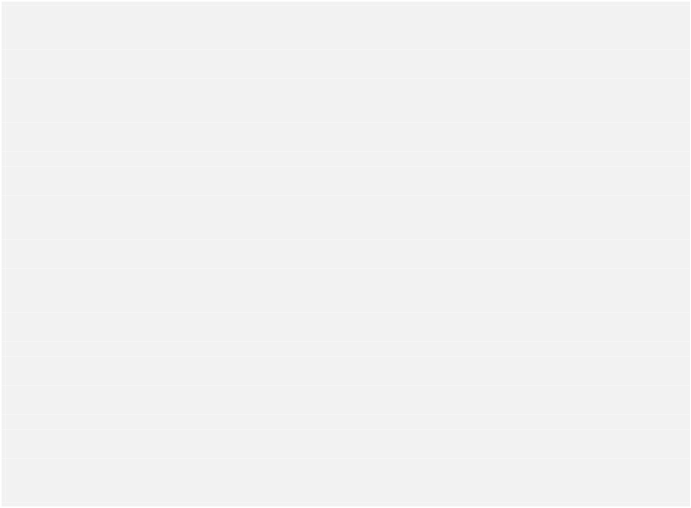
**用法**

valid\_detail ST\_IsValidDetail(geometry geom);

valid\_detail ST\_IsValidDetail(geometry geom, integer ﬂags);

**描述**

返回几何类型是否有效以及无效原因和无效的几何对象是哪些



PostGIS 2.2.0dev开发手册

146 / 686

非常有用的函数，通常用来替代ST\_IsValid 与 ST\_IsValidReason，用来生成无效geometry对象的明细报告

参数flags是一个bit参数，它有如下的值

· 1: 把自我相交的环形成的控当做是有效的，这也被称作 "the ESRI ﬂag". 注意着和OGC 的几何规范是相悖的.

可用版本: 2.0.0 - 要求 GEOS >= 3.3.0.

**样例**

--First 3 Rejects from a successful quintuplet experiment

SELECT gid, reason(ST\_IsValidDetail(the\_geom)), ST\_AsText(location(ST\_IsValidDetail( ←

the\_geom))) as location

FROM

(SELECT ST\_MakePolygon(ST\_ExteriorRing(e.buff), ST\_Accum(f.line)) As the\_geom, gid

FROM (SELECT ST\_Buffer(ST\_MakePoint(x1\*10,y1), z1) As buff, x1\*10 + y1\*100 + z1\*1000 As gid

FROM generate\_series(-4,6) x1

CROSS JOIN generate\_series(2,5) y1

CROSS JOIN generate\_series(1,8) z1

WHERE x1 > y1\*0.5 AND z1 < x1\*y1) As e

INNER JOIN (SELECT ST\_Translate(ST\_ExteriorRing(ST\_Buffer(ST\_MakePoint(x1\*10,y1), z1)),y1 ←

\*1, z1\*2) As line

FROM generate\_series(-3,6) x1

CROSS JOIN generate\_series(2,5) y1

CROSS JOIN generate\_series(1,10) z1

WHERE x1 > y1\*0.75 AND z1 < x1\*y1) As f

ON (ST\_Area(e.buff) > 78 AND ST\_Contains(e.buff, f.line))

GROUP BY gid, e.buff) As quintuplet\_experiment

WHERE ST\_IsValid(the\_geom) = false

ORDER BY gid

LIMIT 3;

gid

|

reason

|

location

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

5330 | Self-intersection | POINT(32 5)

5340 | Self-intersection | POINT(42 5)

5350 | Self-intersection | POINT(52 5)

--simple example

SELECT \* FROM ST\_IsValidDetail(’LINESTRING(220227 150406,2220227 150407,222020 150410)’);

valid | reason | location

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

t

| |

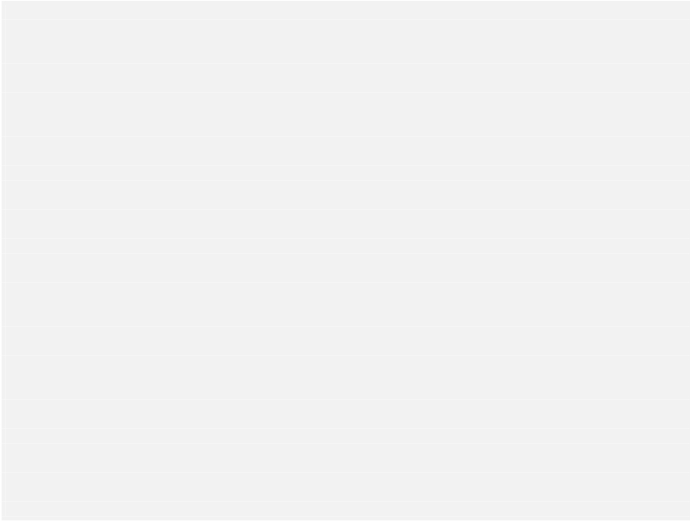
**参考**

ST\_IsValid, ST\_IsValidReason

**8.5.19**

**ST\_M**

ST\_M — 返回一个点的M坐标，如果输入不是point类型，返回NULL。输入值必须是point类型



PostGIS 2.2.0dev开发手册

147 / 686

**用法**

ﬂoat ST\_M(geometry a\_point);

**描述**

返回一个点的M坐标，如果输入不是point类型，返回NULL。输入值必须是point类型

**注意**

该函数还不是OGC SPEC规范中的函数, 但是在这里是为了完善point的坐标抽取函数列表（译者注：除了ST\_M外，还有ST\_X,ST\_Y等等）.

该方法实现了规范 [OpenGIS Simple Features Implementation Speciﬁcation for SQL 1.1.](http://www.opengeospatial.org/standards/sfs)

该方法实现了规范 SQL/MM speciﬁcation.

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

**样例**

SELECT ST\_M(ST\_GeomFromEWKT(’POINT(1 2 3 4)’));

st\_m

------

4

(1 row)

**参考**

ST\_GeomFromEWKT, ST\_X, ST\_Y, ST\_Z

**8.5.20**

**ST\_NDims**

ST\_NDims — 返回一个geometry类型的坐标维度，类型是small int类型，值为2,或4

**用法**

integer ST\_NDims(geometry g1);

**描述**

返回一个geometry类型的坐标维度. PostGIS 支持2 - (x,y) , 3 - (x,y,z) 或 2D 带有测量值m- x,y,m, 以及 4 - 3D

带有测量值m的 x,y,z,m形式

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.











PostGIS 2.2.0dev开发手册

148 / 686

**样例**

SELECT ST\_NDims(ST\_GeomFromText(’POINT(1 1)’)) As d2point,

ST\_NDims(ST\_GeomFromEWKT(’POINT(1 1 2)’)) As d3point,

ST\_NDims(ST\_GeomFromEWKT(’POINTM(1 1 0.5)’)) As d2pointm;

d2point | d3point | d2pointm

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

2 | 3 | 3

**参考**

ST\_CoordDim, ST\_Dimension, ST\_GeomFromEWKT

**8.5.21**

**ST\_NPoints**

ST\_NPoints — 返回geometry的顶点个数.

**用法**

integer ST\_NPoints(geometry g1);

**描述**

返回geometry的顶点个数. 对所有的geometry类型都支持.

提升: 2.0.0版本引入对 Polyhedral Surface的支持

**注意**

在1.3.4版本之前,这个函数如果使用时候含有CURVE类型的几何对象，这个函数会crash崩溃掉，这个bug在1.3.4+后的版本均已经修复

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

该函数支持CircularString和Curve几何类型对象

该函数支持 Polyhedral Surface类型几何对象.

**样例**

SELECT ST\_NPoints(ST\_GeomFromText(’LINESTRING(77.29 29.07,77.42 29.26,77.27 29.31,77.29

29.07)’));

--result

4

--Polygon in 3D space

SELECT ST\_NPoints(ST\_GeomFromEWKT(’LINESTRING(77.29 29.07 1,77.42 29.26 0,77.27 29.31

-1,77.29 29.07 3)’))

--result

4

←

←









PostGIS 2.2.0dev开发手册

149 / 686

**参考**

ST\_NumPoints

**8.5.22**

**ST\_NRings**

ST\_NRings —如果输入的几何类型对象是polygon或者multi-polygon，返回环的个数

**用法**

integer ST\_NRings(geometry geomA);

**描述**

如果输入的几何类型对象是polygon或者multi-polygon，返回环的个数。不像函数NumInteriorRings ，ST\_NRings函数会把外环也统计进去

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

该函数支持CircularString和Curve几何类型对象

**样例**

SELECT ST\_NRings(the\_geom) As Nrings, ST\_NumInteriorRings(the\_geom) As ninterrings

FROM (SELECT ST\_GeomFromText(’POLYGON((1 2, 3 4, 5 6, 1 2))’) As the\_geom) As foo ←

;

nrings | ninterrings

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

1 | 0

(1 row)

**参考**

ST\_NumInteriorRings

**8.5.23**

**ST\_NumGeometries**

ST\_NumGeometries —返回geometry类型是 GEOMETRYCOLLECTION (或 MULTI\*类型)里面包含的几何对象个数，对于单个几何对象，返回1，其他情况返回NULL

**用法**

integer ST\_NumGeometries(geometry geom);





PostGIS 2.2.0dev开发手册

150 / 686

**描述**

返回geometry类型是 GEOMETRYCOLLECTION (或 MULTI\*类型)里面包含的几何对象个数，对于单个几何对象，返回1，其他情况返回NULL

提升: 2.0.0版本支持Polyhedral Surface类型、Triangle类型、TIN类型.

更新: 2.0.0 2.0.0版本之前的版本，如果geometry类型不是collection或/MULTI类型，该函数将会返回NULL值，在2.0.0+版本，对于单个几何对象会返回1，比如POLYGON, LINESTRING, POINT类型

该方法实现了规范 SQL/MM speciﬁcation. SQL-MM 3: 9.1.4

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

该函数支持 Polyhedral Surface类型几何对象.

该函数支持 Triangles 和 Triangulated Irregular Network Surfaces (TIN)类型.

**样例**

--Prior versions would have returned NULL for this -- in 2.0.0 this returns 1

SELECT ST\_NumGeometries(ST\_GeomFromText(’LINESTRING(77.29 29.07,77.42 29.26,77.27

←

29.31,77.29 29.07)’));

--result

1

--Geometry Collection Example - multis count as one geom in a collection

SELECT ST\_NumGeometries(ST\_GeomFromEWKT(’GEOMETRYCOLLECTION(MULTIPOINT(-2 3 , -2 2),

LINESTRING(5 5 ,10 10),

POLYGON((-7 4.2,-7.1 5,-7.1 4.3,-7 4.2)))’));

--result

3

**参考**

ST\_GeometryN, ST\_Multi

**8.5.24**

**ST\_NumInteriorRings**

ST\_NumInteriorRings — 返回输入几何对象里面第一个polygon对象里面的内环数，该函数支持POLYGON 和 MULTIPOLYGON，如果geometry中没有polygon对象，则返回NULL

**用法**

integer ST\_NumInteriorRings(geometry a\_polygon);

**描述**

返回输入几何对象里面第一个polygon对象里面的内环数，该函数支持POLYGON 和 MULTIPOLYGON，如果geometry中没有polygon对象，则返回NULL.

该方法实现了规范 SQL/MM speciﬁcation. SQL-MM 3: 8.2.5











PostGIS 2.2.0dev开发手册

151 / 686

**样例**

--If you have a regular polygon

SELECT gid, field1, field2, ST\_NumInteriorRings(the\_geom) AS numholes

FROM sometable;

--If you have multipolygons

--And you want to know the total number of interior rings in the MULTIPOLYGON

SELECT gid, field1, field2, SUM(ST\_NumInteriorRings(the\_geom)) AS numholes

FROM (SELECT gid, field1, field2, (ST\_Dump(the\_geom)).geom As the\_geom

FROM sometable) As foo

GROUP BY gid, field1,field2;

**参考**

ST\_NumInteriorRing

**8.5.25**

**ST\_NumInteriorRing**

ST\_NumInteriorRing —返回geometry对象里面第一个polygon里面包含的内环数（译者注：这个geometry对象里面包含多个polygon，最大的polygon在外面，该函数就是计算最大的polygon里面含有多少个子polygon） 该函数与ST\_NumInteriorRings等价

**用法**

integer ST\_NumInteriorRing(geometry a\_polygon);

**描述**

返回geometry对象里面第一个polygon里面包含的内环数（译者注：这个geometry对象里面包含多个polygon，最大的polygon在外面，该函数就是计算最大的polygon里面含有多少个子polygon） 该函数与ST\_NumInteriorRings等价. The OpenGIS SPEC规范关于该函数命名有歧义，所以我们同时提供了两种语法

该方法实现了规范 SQL/MM speciﬁcation. SQL-MM 3: 8.2.5

**参考**

ST\_NumInteriorRings

**8.5.26**

**ST\_NumPatches**

ST\_NumPatches — 返回 Polyhedral Surface的面熟. 对于非polyhedral 几何类型对象返回NULL值.

**用法**

integer ST\_NumPatches(geometry g1);



PostGIS 2.2.0dev开发手册

152 / 686

**描述**

返回 Polyhedral Surface的面熟. 对于非polyhedral 几何类型对象返回NULL值. 该函数是MM规范里面函数ST\_NumGeometries的别名. 如果你不在意MM规范的约束，建议使用更快的ST\_NumGeometries

可用版本: 2.0.0

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

该方法实现了规范 [OpenGIS Simple Features Implementation Speciﬁcation for SQL 1.1.](http://www.opengeospatial.org/standards/sfs)

该方法实现了规范 SQL/MM speciﬁcation. SQL-MM 3: ?

该函数支持 Polyhedral Surface类型几何对象.

**样例**

SELECT ST\_NumPatches(ST\_GeomFromEWKT(’POLYHEDRALSURFACE( ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0

0)),

((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),

((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),

((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)) )’));

--result

6

**参考**

ST\_GeomFromEWKT, ST\_NumGeometries

←

**8.5.27**

**ST\_NumPoints**

ST\_NumPoints —返回ST\_LineString 或 ST\_CircularString 值对应的几何对象里面包含的point对象个数.

**用法**

integer ST\_NumPoints(geometry g1);

**描述**

返回ST\_LineString 或 ST\_CircularString 值对应的几何对象里面包含的point对象个数. 1.4版本之前，该函数按照OGC SPEC规范只支持Linestring。从1.4版本开始，该函数是ST\_Npoints的别名函数，而ST\_Npoints这个函数对包括LINESTRING类型的对象返回的是对象的顶点数(译者注：比如一个四边形，顶点就是4个)，建议使用多用途同时支持多类型的ST\_Npoints函数

该方法实现了规范 [OpenGIS Simple Features Implementation Speciﬁcation for SQL 1.1.](http://www.opengeospatial.org/standards/sfs)

该方法实现了规范 SQL/MM speciﬁcation. SQL-MM 3: 7.2.4













PostGIS 2.2.0dev开发手册

153 / 686

**样例**

SELECT ST\_NumPoints(ST\_GeomFromText(’LINESTRING(77.29 29.07,77.42 29.26,77.27 29.31,77.29

29.07)’));

--result

4

**参考**

ST\_NPoints

←

**8.5.28**

**ST\_PatchN**

ST\_PatchN — 返回至少包含一个面的geometry对象的第N个面，支持的输入类型是POLYHEDRALSURFACE, POLYHEDRALSURFACEM. 否则返回NULL值

**用法**

geometry ST\_PatchN(geometry geomA, integer n);

**描述**

返回至少包含一个面的geometry对象的第N个面，支持的输入类型是POLYHEDRALSURFACE, POLYHEDRALSURFACEM. 否则返回NULL值. 对Polyhedral Surface几何类型对象来说，该函数的返回值和ST\_GeometryN 返回值一样，使用ST\_GeometryN 更快

**注意**

几何对象至少包含一个面

**注意**

如果你想得到一个几何对象的所有子几何对象(这里是几何面)，使用ST\_Dump更高效

可用版本: 2.0.0

该方法实现了规范 SQL/MM speciﬁcation. SQL-MM 3: ?

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

该函数支持 Polyhedral Surface类型几何对象.

**样例**











PostGIS 2.2.0dev开发手册

154 / 686

--Extract the 2nd face of the polyhedral surface

SELECT ST\_AsEWKT(ST\_PatchN(geom, 2)) As geomewkt

FROM (

VALUES (ST\_GeomFromEWKT(’POLYHEDRALSURFACE( ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)),

((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),

((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),

((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)) )’)) ) As

foo(geom);

geomewkt

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

POLYGON((0 0 0,0 1 0,1 1 0,1 0 0,0 0 0))

**参考**

ST\_AsEWKT, ST\_GeomFromEWKT, ST\_Dump, ST\_GeometryN, ST\_NumGeometries

←

**8.5.29**

**ST\_PointN**

ST\_PointN —返回单个linestring 或 circular linestring的第N个point对象. 如果几何对象里面不包括LINESTRING对象，则返回NULL

**用法**

geometry ST\_PointN(geometry a\_linestring, integer n);

**描述**

返回linestring 或 circular linestring的包含的第一个LINESTRING的第N个point对象. 如果几何对象里面不包括LINESTRING对象，则返回NULL.

**注意**

OGC SPEC 0.8.0规范开始，下标是从1开始，之前的版本从0开始.

**Note**

如果你想返回multilinestring里面的每一个LINESTRING里面的每一个POINT，使用ST\_Dump

该方法实现了规范 [OpenGIS Simple Features Implementation Speciﬁcation for SQL 1.1.](http://www.opengeospatial.org/standards/sfs)

该方法实现了规范 SQL/MM speciﬁcation. SQL-MM 3: 7.2.5, 7.3.5

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

该函数支持CircularString和Curve几何类型对象













PostGIS 2.2.0dev开发手册

155 / 686

**注意**

更新: 2.0.0 ，该函数不再支持单multilinestring几何对象，在早期的版本，该函数可以很好的支持但line

multilinestring类型，并返回该几何对象的起点。在2.0.0版本，对于其他multistring类型都返回NULL

**样例**

-- Extract all POINTs from a LINESTRING

SELECT ST\_AsText(

ST\_PointN(

column1,

generate\_series(1, ST\_NPoints(column1))

))

FROM ( VALUES (’LINESTRING(0 0, 1 1, 2 2)’::geometry) ) AS foo;

st\_astext

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

POINT(0 0)

POINT(1 1)

POINT(2 2)

(3 rows)

--Example circular string

SELECT ST\_AsText(ST\_PointN(ST\_GeomFromText(’CIRCULARSTRING(1 2, 3 2, 1 2)’),2));

st\_astext

----------

POINT(3 2)

**参考**

ST\_NPoints

**8.5.30**

**ST\_SRID**

ST\_SRID —返回ST\_ Geometry值对应对象的SRID值，该值必须存在spatial\_ref\_sys表中

**用法**

integer ST\_SRID(geometry g1);

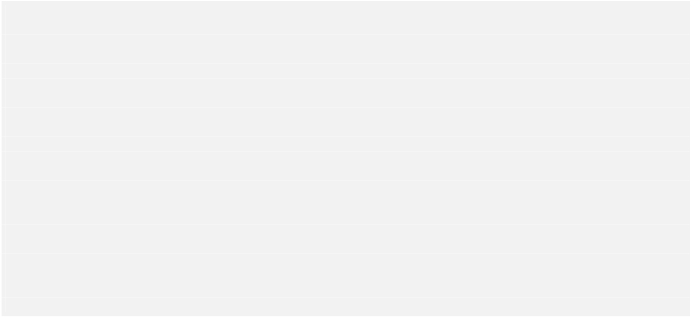
**描述**

返回ST\_ Geometry值对应对象的SRID值，该值必须存在spatial\_ref\_sys表中. 参考 4.3.1节

**注意**

spatial\_ref\_sys table 是一个记录对PostGIS所有已知的空间坐标系统的表，该表可以用来将一个对象的空间坐标系转换到另一种，因此如果你准备对你的几何对象进行坐标系转换时候，请确认SRID







PostGIS 2.2.0dev开发手册

156 / 686

该方法实现了规范 OpenGIS Simple Features Implementation Speciﬁcation for SQL 1.1. s2.1.1.1

该方法实现了规范 SQL/MM speciﬁcation. SQL-MM 3: 5.1.5

该函数支持CircularString和Curve几何类型对象

**样例**

SELECT ST\_SRID(ST\_GeomFromText(’POINT(-71.1043 42.315)’,4326));

--result

4326

**参考**

Section 4.3.1, ST\_GeomFromText, ST\_SetSRID, ST\_Transform

**8.5.31**

**ST\_StartPoint**

ST\_StartPoint — 返回LINESTRING几何对象的起点

**用法**

geometry ST\_StartPoint(geometry geomA);

**描述**

返回LINESTRING几何对象的起点，如果输入类型不是LINESTRING返回NULL

该方法实现了规范 SQL/MM speciﬁcation. SQL-MM 3: 7.1.3

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

**注意**

该函数2.0.0版本更新：不再对单MULTILINESTRING类型支持。在旧版本的PostGIS中，该函数能够很好的支持单MULTILINESTRING类型，并返回起始点，从2.0.0开始，该函数会返回NULL值。旧的方式是不规范的特性，但对于那些把数据存储为MULTILINESTRING的用户来说，在2.0.0版本的PostGIS将会返回NULL值

**样例**

SELECT ST\_AsText(ST\_StartPoint(’LINESTRING(0 1, 0 2)’::geometry));

st\_astext

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

POINT(0 1)

(1 row)

SELECT ST\_StartPoint(’POINT(0 1)’::geometry) IS NULL AS is\_null;

is\_null













PostGIS 2.2.0dev开发手册

157 / 686

----------

t

(1 row)

--3d line

SELECT ST\_AsEWKT(ST\_StartPoint(’LINESTRING(0 1 1, 0 2 2)’::geometry));

st\_asewkt

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

POINT(0 1 1)

(1 row)

**参考**

ST\_EndPoint, ST\_PointN

**8.5.32**

**ST\_Summary**

ST\_Summary —返回geometry对象的文本概要

**用法**

text ST\_Summary(geometry g);

text ST\_Summary(geography g);

**描述**

返回geometry对象的文本概要.

该函数返回的文本描述中，含有中括号的标识意义如下

· M: 有一个 M 坐标

· Z: 有一个 Z 坐标

· B: 有一个缓冲区

· G: 是一个地理坐标

· S: 有空间参考系

可用版本: 1.2.2

提升: 2.0.0 版本添加了对geography对象的支持

提升: 2.1.0 添加了S标识用来表示空间参考坐标系

**样例**

=# SELECT ST\_Summary(ST\_GeomFromText(’LINESTRING(0 0, 1 1)’)) as geom,

ST\_Summary(ST\_GeogFromText(’POLYGON((0 0, 1 1, 1 2, 1 1, 0 0))’)) geog;

geom

|

geog

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

LineString[B] with 2 points | Polygon[BGS] with 1 rings

| ring 0 has 5 points

:

(1 row)

PostGIS 2.2.0dev开发手册

158 / 686

=# SELECT ST\_Summary(ST\_GeogFromText(’LINESTRING(0 0 1, 1 1 1)’)) As geog\_line,

ST\_Summary(ST\_GeomFromText(’SRID=4326;POLYGON((0 0 1, 1 1 2, 1 2 3, 1 1 1, 0 0 1)) ←

’)) As geom\_poly;

;

geog\_line

|

geom\_poly

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

LineString[ZBGS] with 2 points | Polygon[ZBS] with 1 rings

:

ring 0 has 5 points

:

(1 row)

**参考**

PostGIS\_DropBBox, PostGIS\_AddBBox, ST\_Force3DM, ST\_Force3DZ, ST\_Force2D, geography

ST\_IsValid, ST\_IsValid, ST\_IsValidReason, ST\_IsValidDetail

**8.5.33**

**ST\_X**

ST\_X — 返回点的X坐标，如果输入参数不是一个点，返回NULL，输入必须是一个点类型

**用法**

ﬂoat ST\_X(geometry a\_point);

**描述**

返回点的X坐标，如果输入参数不是一个点，返回NULL，输入必须是一个点类型

**注意**

如果你想得到任意几何对象的x坐标的最大和最小值，参考ST\_XMin, ST\_XMax 函数.

该方法实现了规范 SQL/MM speciﬁcation. SQL-MM 3: 6.1.3

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

**样例**

SELECT ST\_X(ST\_GeomFromEWKT(’POINT(1 2 3 4)’));

st\_x

------

1

(1 row)

SELECT ST\_Y(ST\_Centroid(ST\_GeomFromEWKT(’LINESTRING(1 2 3 4, 1 1 1 1)’)));

st\_y

------

1.5

(1 row)







PostGIS 2.2.0dev开发手册

159 / 686

**参考**

ST\_Centroid, ST\_GeomFromEWKT, ST\_M, ST\_XMax, ST\_XMin, ST\_Y, ST\_Z

**8.5.34**

**ST\_XMax**

ST\_XMax — 返回2d或3d的bounding box 或一个几何对象（译者注：bonding box是包含一个区域的最小矩形）的X最大值

**用法**

ﬂoat ST\_XMax(box3d aGeomorBox2DorBox3D);

**描述**

返回2d或3d的bounding box 或一个几何对象（译者注：bonding box是包含一个区域的最小矩形）的X最大值.

**注意**

虽然该函数是为box3d而定义的，但这个函数也支持box2d和geometry类型对象，因为这两种对象可以自动转换成box3d类型。然而你不能用WKT表述的geometry或box2d对象传递进去，因为这种情况下不能做自动转换成box3d类型

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

该函数支持CircularString和Curve几何类型对象

**样例**

SELECT ST\_XMax(’BOX3D(1 2 3, 4 5 6)’);

st\_xmax

-------

4

SELECT ST\_XMax(ST\_GeomFromText(’LINESTRING(1 3 4, 5 6 7)’));

st\_xmax

-------

5

SELECT ST\_XMax(CAST(’BOX(-3 2, 3 4)’ As box2d));

st\_xmax

-------

3

--Observe THIS DOES NOT WORK because it will try to autocast the string representation to a ←

BOX3D

SELECT ST\_XMax(’LINESTRING(1 3, 5 6)’);

--ERROR:

BOX3D parser - doesnt start with BOX3D(

SELECT ST\_XMax(ST\_GeomFromEWKT(’CIRCULARSTRING(220268 150415 1,220227 150505 2,220227

150406 3)’));

st\_xmax

--------

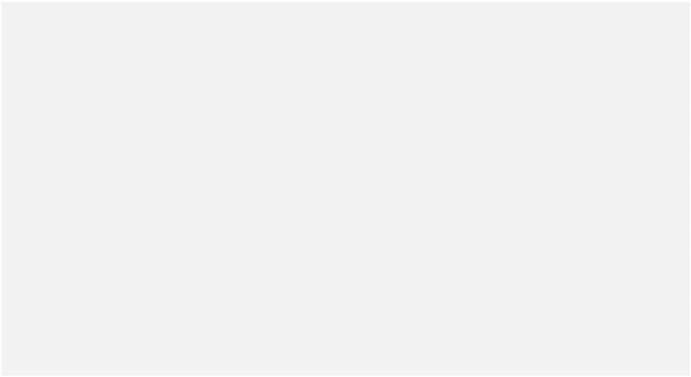
220288.248780547

←









PostGIS 2.2.0dev开发手册

160 / 686

**参考**

ST\_XMin, ST\_YMax, ST\_YMin, ST\_ZMax, ST\_ZMin

**8.5.35**

**ST\_XMin**

ST\_XMin — 返回bounding box或者2d或3d或geometry的X方向范围的最小值

**用法**

ﬂoat ST\_XMin(box3d aGeomorBox2DorBox3D);

**描述**

返回bounding box 或者2d或3d或geometry的X方向范围的最小值.

**注意**

虽然该函数是为box3d而定义的，但这个函数也支持box2d和geometry类型对象，因为这两种对象可以自动转换成box3d类型。然而你不能用WKT表述的geometry或box2d对象传递进去，因为这种情况下不能做自动转换成box3d类型

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

该函数支持CircularString和Curve几何类型对象

**样例**

SELECT ST\_XMin(’BOX3D(1 2 3, 4 5 6)’);

st\_xmin

-------

1

SELECT ST\_XMin(ST\_GeomFromText(’LINESTRING(1 3 4, 5 6 7)’));

st\_xmin

-------

1

SELECT ST\_XMin(CAST(’BOX(-3 2, 3 4)’ As box2d));

st\_xmin

-------

-3

--Observe THIS DOES NOT WORK because it will try to autocast the string representation to a ←

BOX3D

SELECT ST\_XMin(’LINESTRING(1 3, 5 6)’);

--ERROR:

BOX3D parser - doesnt start with BOX3D(

SELECT ST\_XMin(ST\_GeomFromEWKT(’CIRCULARSTRING(220268 150415 1,220227 150505 2,220227

150406 3)’));

st\_xmin

--------

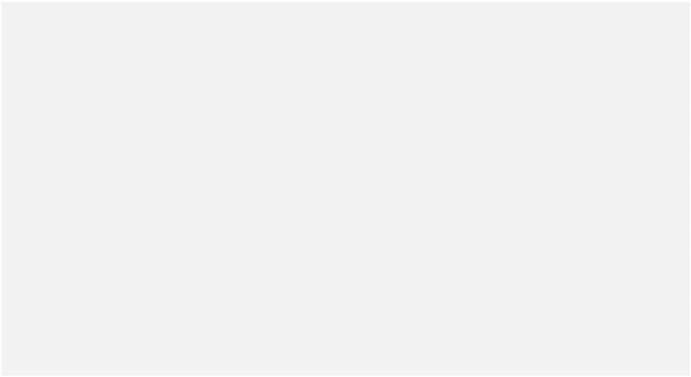
220186.995121892

←









PostGIS 2.2.0dev开发手册

161 / 686

**参考**

ST\_XMax, ST\_YMax, ST\_YMin, ST\_ZMax, ST\_ZMin

**8.5.36**

**ST\_Y**

ST\_Y — 返回输入点的Y坐标，如果输入不是点，返回NULL，输入必须是点类型

**用法**

ﬂoat ST\_Y(geometry a\_point);

**描述**

返回输入点的Y坐标，如果输入不是点，返回NULL，输入必须是点类型.

该方法实现了规范 [OpenGIS Simple Features Implementation Speciﬁcation for SQL 1.1.](http://www.opengeospatial.org/standards/sfs)

该方法实现了规范 SQL/MM speciﬁcation. SQL-MM 3: 6.1.4

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

**样例**

SELECT ST\_Y(ST\_GeomFromEWKT(’POINT(1 2 3 4)’));

st\_y

------

2

(1 row)

SELECT ST\_Y(ST\_Centroid(ST\_GeomFromEWKT(’LINESTRING(1 2 3 4, 1 1 1 1)’)));

st\_y

------

1.5

(1 row)

**参考**

ST\_Centroid, ST\_GeomFromEWKT, ST\_M, ST\_X, ST\_YMax, ST\_YMin, ST\_Z

**8.5.37**

**ST\_YMax**

ST\_YMax —返回bounding box 或者2d或3d或geometry的Y方向范围的最大值.

**用法**

ﬂoat ST\_YMax(box3d aGeomorBox2DorBox3D);







PostGIS 2.2.0dev开发手册

162 / 686

**描述**

返回bounding box 或者2d或3d或geometry的Y方向范围的最大值.

.

**注意**

虽然该函数是为box3d而定义的，但这个函数也支持box2d和geometry类型对象，因为这两种对象可以自动转换成box3d类型。然而你不能用WKT表述的geometry或box2d对象传递进去，因为这种情况下不能做自动转换成box3d类型

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

该函数支持CircularString和Curve几何类型对象

**样例**

SELECT ST\_YMax(’BOX3D(1 2 3, 4 5 6)’);

st\_ymax

-------

5

SELECT ST\_YMax(ST\_GeomFromText(’LINESTRING(1 3 4, 5 6 7)’));

st\_ymax

-------

6

SELECT ST\_YMax(CAST(’BOX(-3 2, 3 4)’ As box2d));

st\_ymax

-------

4

--Observe THIS DOES NOT WORK because it will try to autocast the string representation to a ←

BOX3D

SELECT ST\_YMax(’LINESTRING(1 3, 5 6)’);

--ERROR:

BOX3D parser - doesnt start with BOX3D(

SELECT ST\_YMax(ST\_GeomFromEWKT(’CIRCULARSTRING(220268 150415 1,220227 150505 2,220227

150406 3)’));

st\_ymax

--------

150506.126829327

**参考**

ST\_XMin, ST\_XMax, ST\_YMin, ST\_ZMax, ST\_ZMin

←

**8.5.38**

**ST\_YMin**

ST\_YMin —返回bounding box 或者2d或3d或geometry的Y方向范围的最小值.

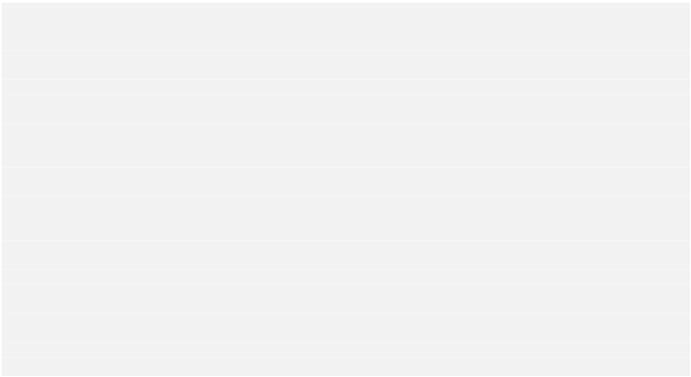
**用法**

ﬂoat ST\_YMin(box3d aGeomorBox2DorBox3D);









PostGIS 2.2.0dev开发手册

163 / 686

**描述**

返回bounding box 或者2d或3d或geometry的Y方向范围的最小值.

**注意**

虽然该函数是为box3d而定义的，但这个函数也支持box2d和geometry类型对象，因为这两种对象可以自动转换成box3d类型。然而你不能用WKT表述的geometry或box2d对象传递进去，因为这种情况下不能做自动转换成box3d类型.

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

该函数支持CircularString和Curve几何类型对象

**样例**

SELECT ST\_YMin(’BOX3D(1 2 3, 4 5 6)’);

st\_ymin

-------

2

SELECT ST\_YMin(ST\_GeomFromText(’LINESTRING(1 3 4, 5 6 7)’));

st\_ymin

-------

3

SELECT ST\_YMin(CAST(’BOX(-3 2, 3 4)’ As box2d));

st\_ymin

-------

2

--Observe THIS DOES NOT WORK because it will try to autocast the string representation to a ←

BOX3D

SELECT ST\_YMin(’LINESTRING(1 3, 5 6)’);

--ERROR:

BOX3D parser - doesnt start with BOX3D(

SELECT ST\_YMin(ST\_GeomFromEWKT(’CIRCULARSTRING(220268 150415 1,220227 150505 2,220227

150406 3)’));

st\_ymin

--------

150406

**参考**

ST\_GeomFromEWKT, ST\_XMin, ST\_XMax, ST\_YMax, ST\_ZMax, ST\_ZMin

←

**8.5.39**

**ST\_Z**

ST\_Z —返回输入点的Z坐标，如果输入不是点，返回NULL，输入必须是点类型

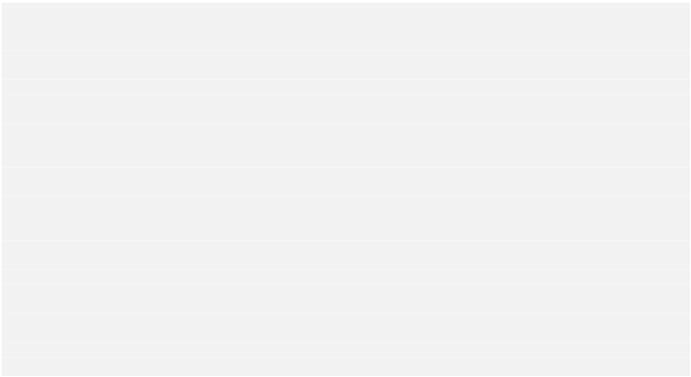
**用法**

ﬂoat ST\_Z(geometry a\_point);









PostGIS 2.2.0dev开发手册

164 / 686

**描述**

返回输入点的Z坐标，如果输入不是点，返回NULL，输入必须是点类型.

该方法实现了规范 SQL/MM speciﬁcation.

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

**样例**

SELECT ST\_Z(ST\_GeomFromEWKT(’POINT(1 2 3 4)’));

st\_z

------

3

(1 row)

**参考**

ST\_GeomFromEWKT, ST\_M, ST\_X, ST\_Y, ST\_ZMax, ST\_ZMin

**8.5.40**

**ST\_ZMax**

ST\_ZMax —返回bounding box 或者2d或3d或geometry的Z方向范围的最大值.

**用法**

ﬂoat ST\_ZMax(box3d aGeomorBox2DorBox3D);

**描述**

返回bounding box 或者2d或3d或geometry的Z方向范围的最大值.

**注意**

虽然该函数是为box3d而定义的，但这个函数也支持box2d和geometry类型对象，因为这两种对象可以自动转换成box3d类型。然而你不能用WKT表述的geometry或box2d对象传递进去，因为这种情况下不能做自动转换成box3d类型.

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

该函数支持CircularString和Curve几何类型对象

**样例**











PostGIS 2.2.0dev开发手册

165 / 686

SELECT ST\_ZMax(’BOX3D(1 2 3, 4 5 6)’);

st\_zmax

-------

6

SELECT ST\_ZMax(ST\_GeomFromEWKT(’LINESTRING(1 3 4, 5 6 7)’));

st\_zmax

-------

7

SELECT ST\_ZMax(’BOX3D(-3 2 1, 3 4 1)’ );

st\_zmax

-------

1

--Observe THIS DOES NOT WORK because it will try to autocast the string representation to a ←

BOX3D

SELECT ST\_ZMax(’LINESTRING(1 3 4, 5 6 7)’);

--ERROR:

BOX3D parser - doesnt start with BOX3D(

SELECT ST\_ZMax(ST\_GeomFromEWKT(’CIRCULARSTRING(220268 150415 1,220227 150505 2,220227

150406 3)’));

st\_zmax

--------

3

**参考**

ST\_GeomFromEWKT, ST\_XMin, ST\_XMax, ST\_YMax, ST\_YMin, ST\_ZMax

←

**8.5.41**

**ST\_Zmﬂag**

ST\_Zmﬂag — 返回一个几何对象的ZM标识，值的对应关系如下，对于2d对象，ZM值为1记为0=2d,以此类推 1=3dm, 2=3dz, 3=4d.

**用法**

smallint ST\_Zmﬂag(geometry geomA);

**描述**

ST\_Zmﬂag — 返回一个几何对象的ZM标识，值的对应关系如下，对于2d对象，ZM值为1记为0=2d,以此类推 1=3dm, 2=3dz, 3=4d..

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

该函数支持CircularString和Curve几何类型对象

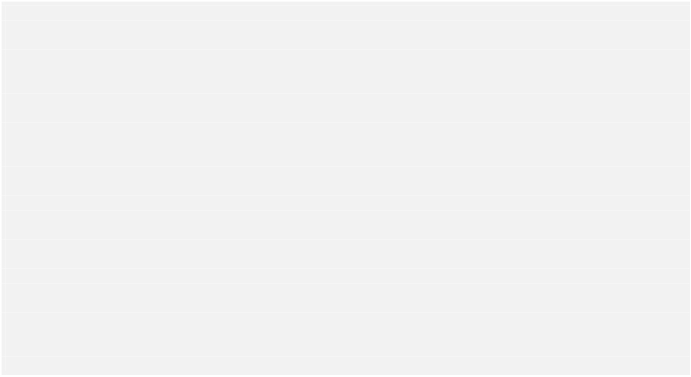
**样例**

SELECT ST\_Zmflag(ST\_GeomFromEWKT(’LINESTRING(1 2, 3 4)’));

st\_zmflag

-----------

0







PostGIS 2.2.0dev 开发手册

166 / 686

SELECT ST\_Zmflag(ST\_GeomFromEWKT(’LINESTRINGM(1 2 3, 3 4 3)’));

st\_zmflag

-----------

1

SELECT ST\_Zmflag(ST\_GeomFromEWKT(’CIRCULARSTRING(1 2 3, 3 4 3, 5 6 3)’));

st\_zmflag

-----------

2

SELECT ST\_Zmflag(ST\_GeomFromEWKT(’POINT(1 2 3 4)’));

st\_zmflag

-----------

3

**请参考**

ST\_CoordDim, ST\_NDims, ST\_Dimension

**8.5.42**

**ST\_ZMin**

ST\_ZMin —返回bounding box 或者2d或3d或geometry的Z方向范围的最小值

**用法**

ﬂoat ST\_ZMin(box3d aGeomorBox2DorBox3D);

**描述**

返回bounding box 或者2d或3d或geometry的Z方向范围的最小值

**注意**

虽然该函数是为box3d而定义的，但这个函数也支持box2d和geometry类型对象，因为这两种对象可以自动转换成box3d类型。然而你不能用WKT表述的geometry或box2d对象传递进去，因为这种情况下不能做自动转换成box3d类型.

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

该函数支持CircularString和Curve几何类型对象

**样例**

SELECT ST\_ZMin(’BOX3D(1 2 3, 4 5 6)’);

st\_zmin

-------

3

SELECT ST\_ZMin(ST\_GeomFromEWKT(’LINESTRING(1 3 4, 5 6 7)’));

st\_zmin

-------

4



PostGIS 2.2.0dev 开发手册

167 / 686

SELECT ST\_ZMin(’BOX3D(-3 2 1, 3 4 1)’ );

st\_zmin

-------

1

--Observe THIS DOES NOT WORK because it will try to autocast the string representation to a ←

BOX3D

SELECT ST\_ZMin(’LINESTRING(1 3 4, 5 6 7)’);

--ERROR:

BOX3D parser - doesnt start with BOX3D(

SELECT ST\_ZMin(ST\_GeomFromEWKT(’CIRCULARSTRING(220268 150415 1,220227 150505 2,220227

150406 3)’));

st\_zmin

--------

1

**请参考**

ST\_GeomFromEWKT, ST\_GeomFromText, ST\_XMin, ST\_XMax, ST\_YMax, ST\_YMin, ST\_ZMax

←

**8.6**

**8.6.1**

**几何对象编辑函数**

**ST\_AddPoint**

ST\_AddPoint — 在LINESTRING对象的某个点的位置之前添加一个点(点的位置计数从0开始)

**用法**

geometry ST\_AddPoint(geometry linestring, geometry point);

geometry ST\_AddPoint(geometry linestring, geometry point, integer position);

**描述**

在LINESTRING对象的某个点的位置之前添加一个点(点的位置计数从0开始)，该参数可以省略或者设置为-1表示在末尾添加点

可用版本: 1.1.0

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

**样例**

--guarantee all linestrings in a table are closed

--by adding the start point of each linestring to the end of the line string

--only for those that are not closed

UPDATE sometable

SET the\_geom = ST\_AddPoint(the\_geom, ST\_StartPoint(the\_geom))

FROM sometable

WHERE ST\_IsClosed(the\_geom) = false;

--Adding point to a 3-d line

SELECT ST\_AsEWKT(ST\_AddPoint(ST\_GeomFromEWKT(’LINESTRING(0 0 1, 1 1 1)’), ST\_MakePoint ←

(1, 2, 3)));



PostGIS 2.2.0dev 开发手册

168 / 686

--result

st\_asewkt

----------

LINESTRING(0 0 1,1 1 1,1 2 3)

**请参考**

ST\_RemovePoint, ST\_SetPoint

**8.6.2**

**ST\_Afﬁne**

ST\_Afﬁne —对一个几何对象在一个步骤中进行3d仿射变换，比如转换、旋转、放大等操作

**用法**

geometry ST\_Afﬁne(geometry geomA, ﬂoat a, ﬂoat b, ﬂoat c, ﬂoat d, ﬂoat e, ﬂoat f, ﬂoat g, ﬂoat h, ﬂoat i, ﬂoat xoff, ﬂoat yoff,

ﬂoat zoff);

geometry ST\_Afﬁne(geometry geomA, ﬂoat a, ﬂoat b, ﬂoat d, ﬂoat e, ﬂoat xoff, ﬂoat yoff);

**描述**

对一个几何对象在一个步骤中进行3d仿射变换，比如转换、旋转、缩放等操作

使用形式 1: 函数调用

ST\_Affine(geom, a, b, c, d, e, f, g, h, i, xoff, yoff, zoff)表示如下的转换矩阵

/

|

|

\

a

d

g

0

b

e

h

0

c

f

i

0

xoff

yoff

zoff

1

\

|

|

/

被转换的对象会做如下形式的转换（译者注：这种变化假如变化后用矩阵用A，变换前是B，仿射矩阵是C，那么这个仿射操作满足A=CB）

x’ = a\*x + b\*y + c\*z + xoff

y’ = d\*x + e\*y + f\*z + yoff

z’ = g\*x + h\*y + i\*z + zoff

所有的转换和缩放函数通过这样的仿射变换来表示

形式 2: 对几何对象进行2d仿射变换.下面的函数调用

ST\_Affine(geom, a, b, d, e, xoff, yoff)

表示转换矩阵

/

|

|

a

d

0

b

e

0

0

0

1

xoff

yoff

0

\

| rsp.

|

/

|

\

a

d

0

b

e

0

xoff

yoff

1

\

|

/

\

0

0

0

1

/

被转换的几何对象，转换后的坐标值表示如下译者注：这种变化假如变化后用矩阵用A，变换前是B，仿射矩阵是C，那么这个仿射操作满足A=CB

x’ = a\*x + b\*y + xoff

y’ = d\*x + e\*y + yoff

z’ = z

PostGIS 2.2.0dev 开发手册

169 / 686

上述的方法的3D方法的在2D维度中的特殊情况

提升: 2.0.0 版本支持Polyhedral Surface类型、Triangle类型、TIN类型.

可用版本: 1.1.2. 从1.2.2版本开始，仿射变换函数Affine改名为ST\_Afﬁne

**注意**

1.3.4版本前,该函数如果包含CURVE类型对象会crash掉，1.3.4+版本均已修复

该函数支持 Polyhedral Surface类型几何对象.

该函数支持 Triangles 和 Triangulated Irregular Network Surfaces (TIN)类型.

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

该函数支持CircularString和Curve几何类型对象

**样例**

--Rotate a 3d line 180 degrees about the z axis.

ST\_Rotate();

注意 this is long-hand for doing

←

SELECT ST\_AsEWKT(ST\_Affine(the\_geom,

cos(pi()), -sin(pi()), 0,

sin(pi()), cos(pi()), 0,

←

0, 0, 1, 0, 0, 0)) As using\_affine,

ST\_AsEWKT(ST\_Rotate(the\_geom, pi())) As using\_rotate

FROM (SELECT ST\_GeomFromEWKT(’LINESTRING(1 2 3, 1 4 3)’) As the\_geom) As foo;

using\_affine

|

using\_rotate

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

LINESTRING(-1 -2 3,-1 -4 3) | LINESTRING(-1 -2 3,-1 -4 3)

(1 row)

--Rotate a 3d line 180 degrees in both the x and z axis

SELECT ST\_AsEWKT(ST\_Affine(the\_geom, cos(pi()), -sin(pi()), 0, sin(pi()), cos(pi()), -sin( ←

pi()), 0, sin(pi()), cos(pi()), 0, 0, 0))

FROM (SELECT ST\_GeomFromEWKT(’LINESTRING(1 2 3, 1 4 3)’) As the\_geom) As foo;

st\_asewkt

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

LINESTRING(-1 -2 -3,-1 -4 -3)

(1 row)

**请参考**

ST\_Rotate, ST\_Scale, ST\_Translate, ST\_TransScale

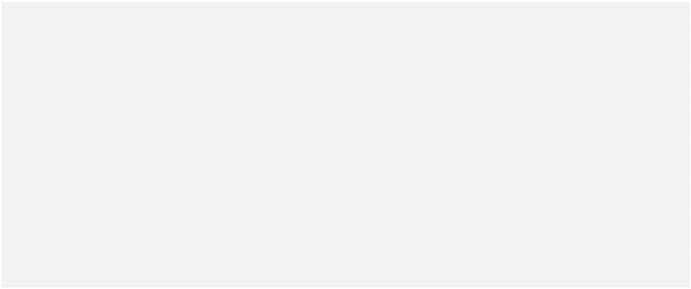
**8.6.3**

**ST\_Force2D**

ST\_Force2D —将一个几何对象转换成2维中的对象，转换后的对象只有X和Y坐标（译者注：其他维度的坐标会被丢弃）

**用法**

geometry ST\_Force2D(geometry geomA);



PostGIS 2.2.0dev 开发手册

170 / 686

**描述**

将一个几何对象转换成2维中的对象，转换后的对象只有X和Y坐标（译者注：其他维度的坐标会被丢弃）.

这个函数对输出必须要与OGC规范兼容的场景很有用(因为当前的OGC规范只规定了2D几何对象)

提升: 2.0.0 版本开始支持Polyhedral Surface几何对象.

变更: 2.1.0. 从 2.0.x 版本开始该函数改名为ST\_Force\_2D.

该函数支持CircularString和Curve几何类型对象

该函数支持 Polyhedral Surface类型几何对象.

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

**样例**

SELECT ST\_AsEWKT(ST\_Force2D(ST\_GeomFromEWKT(’CIRCULARSTRING(1 1 2, 2 3 2, 4 5 2, 6 7 2, 5 6 ←

2)’)));

st\_asewkt

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

CIRCULARSTRING(1 1,2 3,4 5,6 7,5 6)

SELECT

ST\_AsEWKT(ST\_Force2D(’POLYGON((0 0 2,0 5 2,5 0 2,0 0 2),(1 1 2,3 1 2,1 3 2,1 1 2)) ←

’));

st\_asewkt

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

POLYGON((0 0,0 5,5 0,0 0),(1 1,3 1,1 3,1 1))

**请参考**

ST\_Force3D

**8.6.4**

**ST\_Force3D**

ST\_Force3D — 将几何对象转换成XYZ模式，该函数是ST\_Force3DZ别名

**用法**

geometry ST\_Force3D(geometry geomA);

**描述**

将几何对象转换成XYZ模式，该函数是ST\_Force3DZ别名.如果该几何对象没有Z部分坐标，那么Z值被设置为0

提升: 2.0.0 版本开始支持Polyhedral Surface几何对象.

变更: 2.1.0. 从 2.0.x 版本开始该函数改名为ST\_Force\_3D

该函数支持 Polyhedral Surface类型几何对象.



PostGIS 2.2.0dev 开发手册

171 / 686

该函数支持CircularString和Curve几何类型对象

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

**样例**

--Nothing happens to an already 3D geometry

SELECT ST\_AsEWKT(ST\_Force3D(ST\_GeomFromEWKT(’CIRCULARSTRING(1 1 2, 2 3 2, 4 5 2, 6 7 2, ←

5 6 2)’)));

st\_asewkt

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

CIRCULARSTRING(1 1 2,2 3 2,4 5 2,6 7 2,5 6 2)

SELECT

ST\_AsEWKT(ST\_Force3D(’POLYGON((0 0,0 5,5 0,0 0),(1 1,3 1,1 3,1 1))’));

st\_asewkt

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

POLYGON((0 0 0,0 5 0,5 0 0,0 0 0),(1 1 0,3 1 0,1 3 0,1 1 0))

**请参考**

ST\_AsEWKT, ST\_Force2D, ST\_Force3DM, ST\_Force3DZ

**8.6.5**

**ST\_Force3DZ**

ST\_Force3DZ —将几何对象转换成XYZ模式，该函数是ST\_Force3D别名.

**用法**

geometry ST\_Force3DZ(geometry geomA);

**描述**

将几何对象转换成XYZ模式，该函数是ST\_Force3D别名.如果该几何对象没有Z部分坐标，那么Z值被设置为0

版本提升: 2.0.0 版本开始支持Polyhedral Surface几何对象.

版本变更: 2.1.0. 从 2.0.x 版本开始该函数改名为ST\_Force\_3D Z.

该函数支持 Polyhedral Surface类型几何对象.

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

该函数支持CircularString和Curve几何类型对象



PostGIS 2.2.0dev 开发手册

172 / 686

**样例**

--Nothing happens to an already 3D geometry

SELECT ST\_AsEWKT(ST\_Force3DZ(ST\_GeomFromEWKT(’CIRCULARSTRING(1 1 2, 2 3 2, 4 5 2, 6 7 2, 5

6 2)’)));

st\_asewkt

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

CIRCULARSTRING(1 1 2,2 3 2,4 5 2,6 7 2,5 6 2)

←

SELECT

ST\_AsEWKT(ST\_Force3DZ(’POLYGON((0 0,0 5,5 0,0 0),(1 1,3 1,1 3,1 1))’));

st\_asewkt

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

POLYGON((0 0 0,0 5 0,5 0 0,0 0 0),(1 1 0,3 1 0,1 3 0,1 1 0))

**请参考**

ST\_AsEWKT, ST\_Force2D, ST\_Force3DM, ST\_Force3D

**8.6.6**

**ST\_Force3DM**

ST\_Force3DM —将几何对象转换为XYM模式

**用法**

geometry ST\_Force3DM(geometry geomA);

**描述**

将几何对象转换为XYM模式. 如果一个几何对象没有M成分，将会把M值设置为0，如果该几何对象有Z参数值，该值会被丢弃

版本变更: 2.1.0. 2.0.x版本后，该函数改名为ST\_Force\_3DM.

该函数支持CircularString和Curve几何类型对象

**样例**

--Nothing happens to an already 3D geometry

SELECT ST\_AsEWKT(ST\_Force3DM(ST\_GeomFromEWKT(’CIRCULARSTRING(1 1 2, 2 3 2, 4 5 2, 6 7 2, 5

6 2)’)));

st\_asewkt

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

CIRCULARSTRINGM(1 1 0,2 3 0,4 5 0,6 7 0,5 6 0)

←

SELECT

ST\_AsEWKT(ST\_Force3DM(’POLYGON((0 0 1,0 5 1,5 0 1,0 0 1),(1 1 1,3 1 1,1 3 1,1 1 1)) ←

’));

st\_asewkt

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

POLYGONM((0 0 0,0 5 0,5 0 0,0 0 0),(1 1 0,3 1 0,1 3 0,1 1 0))



PostGIS 2.2.0dev 开发手册

173 / 686

**请参考**

ST\_AsEWKT, ST\_Force2D, ST\_Force3DM, ST\_Force3D, ST\_GeomFromEWKT

**8.6.7**

**ST\_Force4D**

ST\_Force4D — 将一个几何对象转换为XYZM模式

**用法**

geometry ST\_Force4D(geometry geomA);

**描述**

将几何对象转换为XYZM模式表述，如果没有Z和M值，则都用0代替

版本变更: 2.1.0. 2.0.x版本后，该函数被改名为ST\_Force\_4D.

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

该函数支持CircularString和Curve几何类型对象

**样例**

--Nothing happens to an already 3D geometry

SELECT ST\_AsEWKT(ST\_Force4D(ST\_GeomFromEWKT(’CIRCULARSTRING(1 1 2, 2 3 2, 4 5 2, 6 7 2, 5 6 ←

2)’)));

st\_asewkt

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

CIRCULARSTRING(1 1 2 0,2 3 2 0,4 5 2 0,6 7 2 0,5 6 2 0)

SELECT

ST\_AsEWKT(ST\_Force4D(’MULTILINESTRINGM((0 0 1,0 5 2,5 0 3,0 0 4),(1 1 1,3 1 1,1 3

←

1,1 1 1))’));

st\_asewkt

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

MULTILINESTRING((0 0 0 1,0 5 0 2,5 0 0 3,0 0 0 4),(1 1 0 1,3 1 0 1,1 3 0 1,1 1 0 1))

**请参考**

ST\_AsEWKT, ST\_Force2D, ST\_Force3DM, ST\_Force3D

**8.6.8**

**ST\_ForceCollection**

ST\_ForceCollection — 把geometry对象转换成GEOMETRYCOLLECTION类型对象

**用法**

geometry ST\_ForceCollection(geometry geomA);



PostGIS 2.2.0dev 开发手册

174 / 686

**描述**

把geometry对象转换成GEOMETRYCOLLECTION类型对象. 对于简化WKB表述方式很有用

版本提升: 2.0.0 版本开始支持Polyhedral Surface几何对象.

可用版本: 1.2.2, 1.3.4版本之前该函数处理Curve类型对象会crash掉， 1.3.4+后版本已经修复

版本变更: 2.1.0. 2.0.x后该函数改名为ST\_Force\_Collection.

该函数支持 Polyhedral Surface类型几何对象.

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

该函数支持CircularString和Curve几何类型对象

**样例**

SELECT

ST\_AsEWKT(ST\_ForceCollection(’POLYGON((0 0 1,0 5 1,5 0 1,0 0 1),(1 1 1,3 1 1,1 3

←

1,1 1 1))’));

st\_asewkt

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

GEOMETRYCOLLECTION(POLYGON((0 0 1,0 5 1,5 0 1,0 0 1),(1 1 1,3 1 1,1 3 1,1 1 1)))

SELECT ST\_AsText(ST\_ForceCollection(’CIRCULARSTRING(220227 150406,2220227 150407,220227

150406)’));

st\_astext

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

GEOMETRYCOLLECTION(CIRCULARSTRING(220227 150406,2220227 150407,220227 150406))

(1 row)

-- POLYHEDRAL example --

SELECT ST\_AsEWKT(ST\_ForceCollection(’POLYHEDRALSURFACE(((0 0 0,0 0 1,0 1 1,0 1 0,0 0 0)),

((0 0 0,0 1 0,1 1 0,1 0 0,0 0 0)),

((0 0 0,1 0 0,1 0 1,0 0 1,0 0 0)),

((1 1 0,1 1 1,1 0 1,1 0 0,1 1 0)),

((0 1 0,0 1 1,1 1 1,1 1 0,0 1 0)),

((0 0 1,1 0 1,1 1 1,0 1 1,0 0 1)))’))

st\_asewkt

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

GEOMETRYCOLLECTION(

POLYGON((0 0 0,0 0 1,0 1 1,0 1 0,0 0 0)),

POLYGON((0 0 0,0 1 0,1 1 0,1 0 0,0 0 0)),

POLYGON((0 0 0,1 0 0,1 0 1,0 0 1,0 0 0)),

POLYGON((1 1 0,1 1 1,1 0 1,1 0 0,1 1 0)),

POLYGON((0 1 0,0 1 1,1 1 1,1 1 0,0 1 0)),

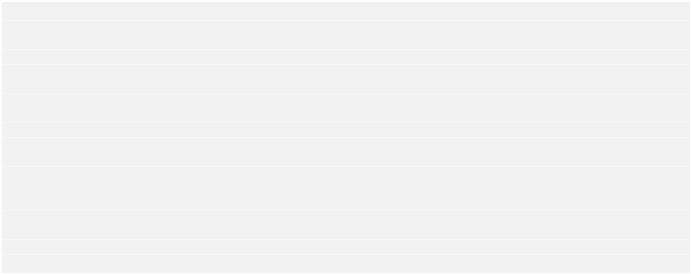
POLYGON((0 0 1,1 0 1,1 1 1,0 1 1,0 0 1))

)

**请参考**

ST\_AsEWKT, ST\_Force2D, ST\_Force3DM, ST\_Force3D, ST\_GeomFromEWKT

←



PostGIS 2.2.0dev 开发手册

175 / 686

**8.6.9**

**ST\_ForceSFS**

ST\_ForceSFS — 把几何对象转换成符合SFS 1.1版本规范的几何类型对象

**用法**

geometry ST\_ForceSFS(geometry geomA);

geometry ST\_ForceSFS(geometry geomA, text version);

**描述**

该函数支持 Polyhedral Surface类型几何对象.

该函数支持 Triangles 和 Triangulated Irregular Network Surfaces (TIN)类型.

该函数支持CircularString和Curve几何类型对象

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

**8.6.10**

**ST\_ForceRHR**

ST\_ForceRHR —按照右手坐标定则对polygon的顶点进行转向

**用法**

boolean ST\_ForceRHR(geometry g);

**描述**

按照右手坐标定则对polygon的顶点进行转向.在GIS术语中，这意味着被多边形包围的区域会被转向边界的右方。特别地：外环是顺时针方向的，而内环是逆时针方向的

版本提升: 2.0.0 版本开始支持Polyhedral Surface几何对象.

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

该函数支持 Polyhedral Surface类型几何对象.

**样例**

SELECT ST\_AsEWKT(

ST\_ForceRHR(

’POLYGON((0 0 2, 5 0 2, 0 5 2, 0 0 2),(1 1 2, 1 3 2, 3 1 2, 1 1 2))’

)

);

st\_asewkt

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

POLYGON((0 0 2,0 5 2,5 0 2,0 0 2),(1 1 2,3 1 2,1 3 2,1 1 2))

(1 row)



PostGIS 2.2.0dev 开发手册

176 / 686

**请参考**

ST\_BuildArea, ST\_Polygonize, ST\_Reverse

**8.6.11**

**ST\_ForceCurve**

ST\_ForceCurve — 把一个geometry对象向上转型为对应的curve 类型，如果可以的话(译者注：向上转型和Java里面的向上转型类似)

**用法**

geometry ST\_ForceCurve(geometry g);

**描述**

把一个geometry对象向上转型为对应的curve 类型，如果可以的话(译者注：向上转型和Java里面的向上转型类似): line向上转型后是compoundcurve类型, multiline类型向上转型是multicurve，polygon向上转型是curvepolygon， multipolygon向上转型是multisurface.

可用版本: 2.2.0

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

**样例**

SELECT ST\_AsText(

ST\_ForceCurve(

’POLYGON((0 0 2, 5 0 2, 0 5 2, 0 0 2),(1 1 2, 1 3 2, 3 1 2, 1 1 2))’

)

);

st\_astext

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

CURVEPOLYGON Z ((0 0 2,5 0 2,0 5 2,0 0 2),(1 1 2,1 3 2,3 1 2,1 1 2))

(1 row)

**请参考**

ST\_LineToCurve

**8.6.12**

**ST\_LineMerge**

ST\_LineMerge — 把一些LineString对象组合在一起，形成一个MULTILINESTRING对象

**用法**

geometry ST\_LineMerge(geometry amultilinestring);



PostGIS 2.2.0dev 开发手册

177 / 686

**描述**

把一些LineString对象组合在一起，形成一个MULTILINESTRING对象.

**注意**

该函数值支持MULTILINESTRING/LINESTRING类型对象.如果把该函数应用于polygon 或 geometry collection 类型对象,该函数将返回一个空的GEOMETRYCOLLECTION对象

可用版本: 1.1.0

**注意**

版本要求 GEOS >= 2.1.0

**样例**

SELECT ST\_AsText(ST\_LineMerge(

ST\_GeomFromText(’MULTILINESTRING((-29 -27,-30 -29.7,-36 -31,-45 -33),(-45 -33,-46 -32))’)

)

);

st\_astext

-------------------------------------------------------------------------------------------------- ←

LINESTRING(-29 -27,-30 -29.7,-36 -31,-45 -33,-46 -32)

(1 row)

--If can’t be merged - original MULTILINESTRING is returned

SELECT ST\_AsText(ST\_LineMerge(

ST\_GeomFromText(’MULTILINESTRING((-29 -27,-30 -29.7,-36 -31,-45 -33),(-45.2 -33.2,-46 -32)) ←

’)

)

);

st\_astext

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

MULTILINESTRING((-45.2 -33.2,-46 -32),(-29 -27,-30 -29.7,-36 -31,-45 -33))

**请参考**

ST\_Segmentize, ST\_LineSubstring

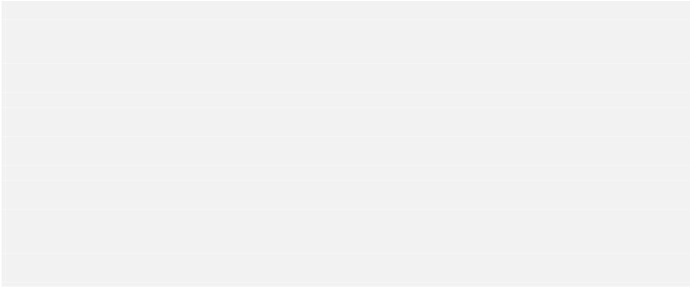
**8.6.13**

**ST\_CollectionExtract**

ST\_CollectionExtract — 根据输入的(multi)几何对象，返回一个(multi)几何对象，返回的几何对象只包含指定的类型的元素

**用法**

geometry ST\_CollectionExtract(geometry collection, integer type);



PostGIS 2.2.0dev 开发手册

178 / 686

**描述**

根据输入的(multi)几何对象，返回一个(multi)几何对象，返回的几何对象只包含指定的类型的元素.没有指定的类型子几何对象会被忽略掉.如果没有对应类型的子几何对象，该函数会返回空几何对象。本函数只支持point、 line和 polygon类型。类型值和类型名称对应关系如下：1 == POINT, 2 == LINESTRING, 3 == POLYGON.

可用版本: 1.5.0

**注意**

1.5.3版本之前，这个函数无论输入什么类型，返回的都是non-collection. 1.5.3版本中，不匹配的单几何对象返回NULL值，在2.0.0及之后的版本，每个不匹配的输入参数type类型值都返回空类型

**警告**

当指定type类型为3的时候，返回值为 multipolygon对象，即使polygon的边界是共享的，在许多情况下，比如将该函数作用于一个ST\_Split值对象时候，返回值是一个无效的multipolygon对象

**样例**

-- Constants: 1 == POINT, 2 == LINESTRING, 3 == POLYGON

SELECT ST\_AsText(ST\_CollectionExtract(ST\_GeomFromText(’GEOMETRYCOLLECTION( ←

GEOMETRYCOLLECTION(POINT(0 0)))’),1));

st\_astext

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

MULTIPOINT(0 0)

(1 row)

SELECT ST\_AsText(ST\_CollectionExtract(ST\_GeomFromText(’GEOMETRYCOLLECTION( ←

GEOMETRYCOLLECTION(LINESTRING(0 0, 1 1)),LINESTRING(2 2, 3 3))’),2));

st\_astext

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

MULTILINESTRING((0 0, 1 1), (2 2, 3 3))

(1 row)

**请参考**

ST\_Multi, ST\_Dump, ST\_CollectionHomogenize

**8.6.14**

**ST\_CollectionHomogenize**

ST\_CollectionHomogenize —根据给定的geometry collection对象，返回一个最简化的对象

**用法**

geometry ST\_CollectionHomogenize(geometry collection);

**描述**

根据给定的geometry collection对象，返回一个最简化的对象。单例对象返回的还是单例对象，同类的collection对象会返回一个合适的multi-type类型



PostGIS 2.2.0dev 开发手册

179 / 686

**警告**

当指定type类型为3的时候，返回值为 multipolygon对象，即使polygon的边界是共享的，在许多情况下，比如将该函数作用于一个ST\_Split值对象时候，返回值是一个无效的multipolygon对象

可用版本: 2.0.0

**样例**

SELECT ST\_AsText(ST\_CollectionHomogenize(’GEOMETRYCOLLECTION(POINT(0 0))’));

st\_astext

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

POINT(0 0)

(1 row)

SELECT ST\_AsText(ST\_CollectionHomogenize(’GEOMETRYCOLLECTION(POINT(0 0),POINT(1 1))’));

st\_astext

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

MULTIPOINT(0 0,1 1)

(1 row)

**请参考**

ST\_Multi, ST\_CollectionExtract

**8.6.15**

**ST\_Multi**

ST\_Multi — 返回几何对象为一个MULTI\* 类型几何对象，如果几何对象本身就是MULTI\*类型的，返回值不变，即和输入一样.

**用法**

geometry ST\_Multi(geometry g1);

**描述**

返回几何对象为一个MULTI\* 类型几何对象，如果几何对象本身就是MULTI\*类型的，返回值不变，即和输入值一样.

**样例**

SELECT ST\_AsText(ST\_Multi(ST\_GeomFromText(’POLYGON((743238 2967416,743238 2967450,

743265 2967450,743265.625 2967416,743238 2967416))’)));

st\_astext

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

MULTIPOLYGON(((743238 2967416,743238 2967450,743265 2967450,743265.625 2967416,

743238 2967416)))

(1 row)



PostGIS 2.2.0dev 开发手册

180 / 686

**请参考**

ST\_AsText

**8.6.16**

**ST\_RemovePoint**

ST\_RemovePoint — 从一个LINESTRING对象中移除一个Point点，下标从0开始

**用法**

geometry ST\_RemovePoint(geometry linestring, integer offset);

**描述**

从一个LINESTRING对象中移除一个Point点，下标从0开始.这个函数对于把一个封闭的环拆成一个开放的LINESTRING很有用

可用版本: 1.1.0

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

**样例**

--guarantee no LINESTRINGS are closed

--by removing the end point.

The below assumes the\_geom is of type LINESTRING

UPDATE sometable

SET the\_geom = ST\_RemovePoint(the\_geom, ST\_NPoints(the\_geom) - 1)

FROM sometable

WHERE ST\_IsClosed(the\_geom) = true;

**请参考**

ST\_AddPoint, ST\_NPoints, ST\_NumPoints

**8.6.17**

**ST\_Reverse**

ST\_Reverse —按顶点的逆序返回一个geometry对象

**用法**

geometry ST\_Reverse(geometry g1);

**描述**

该函数支持任意几何类型对象，按顶点的逆序返回一个geometry对象



PostGIS 2.2.0dev 开发手册

181 / 686

**样例**

SELECT ST\_AsText(the\_geom) as line, ST\_AsText(ST\_Reverse(the\_geom)) As reverseline

FROM

(SELECT ST\_MakeLine(ST\_MakePoint(1,2),

ST\_MakePoint(1,10)) As the\_geom) as foo;

--result

line

|

reverseline

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

LINESTRING(1 2,1 10) | LINESTRING(1 10,1 2)

**8.6.18**

**ST\_Rotate**

ST\_Rotate — 返回一个几何对象以某个点为中心点，逆时针旋转指定弧度后的对象.

**用法**

geometry ST\_Rotate(geometry geomA, ﬂoat rotRadians);

geometry ST\_Rotate(geometry geomA, ﬂoat rotRadians, ﬂoat x0, ﬂoat y0);

geometry ST\_Rotate(geometry geomA, ﬂoat rotRadians, geometry pointOrigin);

**描述**

返回一个几何对象以原点为中心点，逆时针旋转指定弧度后的对象. 旋转的中心点可以指定为Point几何对象的格式，或者x，y坐标的格式，如果旋转中心点没有指定，默认是按照POINT(0,0)即原点为中心点。

版本提升: 2.0.0 版本支持Polyhedral Surface类型、Triangle类型、TIN类型.

版本提升: 2.0.0 additional parameters for specifying the origin of rotation were added.

可用版本: 1.1.2. Name changed from Rotate to ST\_Rotate in 1.2.2

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

该函数支持CircularString和Curve几何类型对象

该函数支持 Polyhedral Surface类型几何对象.

该函数支持 Triangles 和 Triangulated Irregular Network Surfaces (TIN)类型.

**样例**

--Rotate 180 degrees

SELECT ST\_AsEWKT(ST\_Rotate(’LINESTRING (50 160, 50 50, 100 50)’, pi()));

st\_asewkt

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

LINESTRING(-50 -160,-50 -50,-100 -50)

(1 row)

--Rotate 30 degrees counter-clockwise at x=50, y=160

SELECT ST\_AsEWKT(ST\_Rotate(’LINESTRING (50 160, 50 50, 100 50)’, pi()/6, 50, 160));

st\_asewkt

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

LINESTRING(50 160,105 64.7372055837117,148.301270189222 89.7372055837117)



PostGIS 2.2.0dev 开发手册

182 / 686

(1 row)

--Rotate 60 degrees clockwise from centroid

SELECT ST\_AsEWKT(ST\_Rotate(geom, -pi()/3, ST\_Centroid(geom)))

FROM (SELECT ’LINESTRING (50 160, 50 50, 100 50)’::geometry AS geom) AS foo;

st\_asewkt

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

LINESTRING(116.4225 130.6721,21.1597 75.6721,46.1597 32.3708)

(1 row)

**请参考**

ST\_Afﬁne, ST\_RotateX, ST\_RotateY, ST\_RotateZ

**8.6.19**

**ST\_RotateX**

ST\_RotateX — 将一个几何对象绕X轴旋转指定弧度

**用法**

geometry ST\_RotateX(geometry geomA, ﬂoat rotRadians);

**描述**

将一个几何对象绕X轴旋转指定弧度.

**注意**

ST\_RotateX(geomA, rotRadians)是函数ST\_Affine(geomA, 1, 0, 0, 0, cos(rotRadians), -sin(rotRadians), 0, sin(rotRadians), cos(rotRadians), 0, 0, 0)的简化版（译者注：仿射函数ST\_Affine参数太多了，这个函数是仿射函数的一种特殊情况的简化版，相比较而言更简单一些）

版本提升: 2.0.0 版本支持Polyhedral Surface类型、Triangle类型、TIN类型.

可用版本: 1.1.2. 在1.22版本该函数名称由RotateX 改为 ST\_RotateX

该函数支持 Polyhedral Surface类型几何对象.

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

该函数支持 Triangles 和 Triangulated Irregular Network Surfaces (TIN)类型.

**样例**

--Rotate a line 90 degrees along x-axis

SELECT ST\_AsEWKT(ST\_RotateX(ST\_GeomFromEWKT(’LINESTRING(1 2 3, 1 1 1)’), pi()/2));

st\_asewkt

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

LINESTRING(1 -3 2,1 -1 1)



PostGIS 2.2.0dev 开发手册

183 / 686

**请参考**

ST\_Afﬁne, ST\_RotateY, ST\_RotateZ

**8.6.20**

**ST\_RotateY**

ST\_RotateY —将一个几何对象绕Y轴旋转指定弧度

**用法**

geometry ST\_RotateY(geometry geomA, ﬂoat rotRadians);

**描述**

将一个几何对象绕Y轴旋转指定弧度.

**注意**

ST\_RotateY(geomA, rotRadians)是函数ST\_Affine(geomA, cos(rotRadians), 0,sin(rotRadians), 0, 1, 0, -sin(rotRadians), 0, cos(rotRadians), 0, 0, 0)的简化版（译者注：仿射函数ST\_Affine参数太多了，这个函数是仿射函数的一种特殊情况的简化版，相比较而言更简单一些）

可用版本: 1.1.2.在1.22版本该函数名称由RotateY改为 ST\_RotateY

版本提升: 2.0.0 版本支持Polyhedral Surface类型、Triangle类型、TIN类型.

该函数支持 Polyhedral Surface类型几何对象.

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

该函数支持 Triangles 和 Triangulated Irregular Network Surfaces (TIN)类型.

**样例**

--Rotate a line 90 degrees along y-axis

SELECT ST\_AsEWKT(ST\_RotateY(ST\_GeomFromEWKT(’LINESTRING(1 2 3, 1 1 1)’), pi()/2));

st\_asewkt

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

LINESTRING(3 2 -1,1 1 -1)

**请参考**

ST\_Afﬁne, ST\_RotateX, ST\_RotateZ

**8.6.21**

**ST\_RotateZ**

ST\_RotateZ —将一个几何对象绕Z轴旋转指定弧度

**用法**

geometry ST\_RotateZ(geometry geomA, ﬂoat rotRadians);



PostGIS 2.2.0dev 开发手册

184 / 686

**描述**

Rotate a geometry geomA - rotRadians about the Z axis.

**注意**

该函数与ST\_Rotate等效

**注意**

ST\_RotateZ(geomA, rotRadians)是函数ST\_Affine(geomA, cos(rotRadians), -sin(rotRadians), 0, sin(rotRadians), cos(rotRadians), 0, 0, 0, 1, 0, 0,0)的简化版（译者注：仿射函数ST\_Affine参数太多了，这个函数是仿射函数的一种特殊情况的简化版，相比较而言更简单一些）

版本提升: 2.0.0 版本支持Polyhedral Surface类型、Triangle类型、TIN类型.

可用版本: 1.1.2. Name changed from RotateZ to ST\_RotateZ in 1.2.2

**注意**

1.3.4版本前,该函数如果包含CURVE类型对象会crash掉，1.3.4+版本均已修复

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

该函数支持CircularString和Curve几何类型对象

该函数支持 Polyhedral Surface类型几何对象.

该函数支持 Triangles 和 Triangulated Irregular Network Surfaces (TIN)类型.

**样例**

--Rotate a line 90 degrees along z-axis

SELECT ST\_AsEWKT(ST\_RotateZ(ST\_GeomFromEWKT(’LINESTRING(1 2 3, 1 1 1)’), pi()/2));

st\_asewkt

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

LINESTRING(-2 1 3,-1 1 1)

--Rotate a curved circle around z-axis

SELECT ST\_AsEWKT(ST\_RotateZ(the\_geom, pi()/2))

FROM (SELECT ST\_LineToCurve(ST\_Buffer(ST\_GeomFromText(’POINT(234 567)’), 3)) As the\_geom)

As foo;

st\_asewkt

←

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

CURVEPOLYGON(CIRCULARSTRING(-567 237,-564.87867965644 236.12132034356,-564

234,-569.12132034356 231.87867965644,-567 237))

←



PostGIS 2.2.0dev 开发手册

185 / 686

**请参考**

ST\_Afﬁne, ST\_RotateX, ST\_RotateY

**8.6.22**

**ST\_Scale**

ST\_Scale —通过将输入对象的坐标乘以一个系数来对对象进行缩放(缩小和放大)，例如ST\_Scale(geom, Xfactor,Yfactor, Zfactor).

**用法**

geometry ST\_Scale(geometry geomA, ﬂoat XFactor, ﬂoat YFactor, ﬂoat ZFactor);

geometry ST\_Scale(geometry geomA, ﬂoat XFactor, ﬂoat YFactor);

**描述**

通过将输入对象的坐标乘以一个系数来对对象进行缩放(缩小和放大)，例如ST\_Scale(geom, Xfactor,Yfactor, Zfactor).

**注意**

ST\_Scale(geomA, XFactor, YFactor, ZFactor)是函数ST\_Affine(geomA, XFactor, 0, 0, 0, YFactor, 0, 0, 0, ZFactor, 0, 0, 0)的简化版（译者注：仿射函数ST\_Affine参数太多了，这个函数是仿射函数的一种特殊情况的简化版，相比较而言更简单一些）

**注意**

1.3.4版本前,该函数如果包含CURVE类型对象会crash掉，1.3.4+版本均已修复

可用版本: 1.1.0.

版本提升: 2.0.0 版本支持Polyhedral Surface类型、Triangle类型、TIN类型.

该函数支持 Polyhedral Surface类型几何对象.

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

该函数支持CircularString和Curve几何类型对象

该函数支持 Triangles 和 Triangulated Irregular Network Surfaces (TIN)类型.

**样例**

--Version 1: scale X, Y, Z

SELECT ST\_AsEWKT(ST\_Scale(ST\_GeomFromEWKT(’LINESTRING(1 2 3, 1 1 1)’), 0.5, 0.75, 0.8));

st\_asewkt

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

LINESTRING(0.5 1.5 2.4,0.5 0.75 0.8)

--Version 2: Scale X Y

SELECT ST\_AsEWKT(ST\_Scale(ST\_GeomFromEWKT(’LINESTRING(1 2 3, 1 1 1)’), 0.5, 0.75));



PostGIS 2.2.0dev 开发手册

186 / 686

st\_asewkt

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

LINESTRING(0.5 1.5 3,0.5 0.75 1)

**请参考**

ST\_Afﬁne, ST\_TransScale

**8.6.23**

**ST\_Segmentize**

ST\_Segmentize —返回一个修改后的geometry对象，让这个对象的的每一段的长度不能超过给出的最大长度。只在2D维度能进行计算。对于几何对象来说，长度单位需要在SRS表中，对于geography地理对象来说，距离单位是米

译者注：简单的说，这个函数可以把一个比较长的线段给切分，让最长的线段不超过给定的长度.

**用法**

geometry ST\_Segmentize(geometry geom, ﬂoat max\_segment\_length);

geometry ST\_Segmentize(geography geog, ﬂoat max\_segment\_length);

**描述**

返回一个修改后的geometry对象，让这个对象的的每一段的长度不能超过给出的最大长度。只在2D维度能进行计算。对于几何对象来说，长度单位需要在SRS表中，对于geography地理对象来说，距离单位是米

可用版本: 1.2.2

版本提升: 2.1.0引入支持geography对象.

版本变更: 2.1.0 引入对geography类型对象支持:该查询 SELECT ST\_Segmentize(’LINESTRING(1 2, 3 4)’,0.5);会返回函数名歧义的错误。你需要声明正确的对象类型，例如一个geometry或geography类型的列, 使用函数ST\_GeomFromText, ST\_GeogFromText 或 SELECT ST\_Segmentize(’LINESTRING(1 2, 3 4)’::geometry,0.5);查询

**注意**

该函数会增加几何对象的片段数，并且片段的长度不会比最大的长度限制长。

**样例**

SELECT ST\_AsText(ST\_Segmentize(

ST\_GeomFromText(’MULTILINESTRING((-29 -27,-30 -29.7,-36 -31,-45 -33),(-45 -33,-46 -32))’)

,5)

);

st\_astext

-------------------------------------------------------------------------------------------------- ←

MULTILINESTRING((-29 -27,-30 -29.7,-34.886615700134 -30.758766735029,-36 -31,

-40.8809353009198 -32.0846522890933,-45 -33),

(-45 -33,-46 -32))

(1 row)

SELECT ST\_AsText(ST\_Segmentize(ST\_GeomFromText(’POLYGON((-29 28, -30 40, -29 28))’),10));

st\_astext



PostGIS 2.2.0dev 开发手册

187 / 686

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

POLYGON((-29 28,-29.8304547985374 37.9654575824488,-30 40,-29.1695452014626

30.0345424175512,-29 28))

(1 row)

**请参考**

ST\_LineSubstring

←

**8.6.24**

**ST\_SetPoint**

ST\_SetPoint —用给定的点替换一个LINESTRING对象的第N个点，下标从0开始

**用法**

geometry ST\_SetPoint(geometry linestring, integer zerobasedposition, geometry point);

**描述**

用给定的点替换一个LINESTRING对象的第N个点，下标从0开始. 在一个几何对象的顶点移动时候，为了保持和其他点连接关系，会触发新的连接关系，这一点很有用

可用版本: 1.1.0

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

**样例**

--Change first point in line string from -1 3 to -1 1

SELECT ST\_AsText(ST\_SetPoint(’LINESTRING(-1 2,-1 3)’, 0, ’POINT(-1 1)’));

st\_astext

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

LINESTRING(-1 1,-1 3)

---Change last point in a line string (lets play with 3d linestring this time)

SELECT ST\_AsEWKT(ST\_SetPoint(foo.the\_geom, ST\_NumPoints(foo.the\_geom) - 1, ST\_GeomFromEWKT ←

(’POINT(-1 1 3)’)))

FROM (SELECT ST\_GeomFromEWKT(’LINESTRING(-1 2 3,-1 3 4, 5 6 7)’) As the\_geom) As foo;

st\_asewkt

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

LINESTRING(-1 2 3,-1 3 4,-1 1 3)

**请参考**

ST\_AddPoint, ST\_NPoints, ST\_NumPoints, ST\_PointN, ST\_RemovePoint

**8.6.25**

**ST\_SetSRID**

ST\_SetSRID —把一个几何对象设置为一个给定的整型值.



PostGIS 2.2.0dev 开发手册

188 / 686

**用法**

geometry ST\_SetSRID(geometry geom, integer srid);

**描述**

把一个几何对象设置为一个给定的整型值. 在查询时候，构建bounding box很有用（译者注：该术语arcgis给的翻译是最小边界框，读者可以根据自己理解即可，翻译出来可能会有一些怪怪的，当做一个术语可能更好理解）

**注意**

该函数不做任何方式的坐标转换，它只是把该对象的空间参考系的SRID元数据信息。你可以使用函数ST\_Transform来把指定的几何对象投影到新的坐标系统中

该函数方法实现了规范 [OpenGIS Simple Features Implementation Speciﬁcation for SQL 1.1.](http://www.opengeospatial.org/standards/sfs)

该函数支持CircularString和Curve几何类型对象

**样例**

-- Mark a point as WGS 84 long lat --

SELECT ST\_SetSRID(ST\_Point(-123.365556, 48.428611),4326) As wgs84long\_lat;

-- the ewkt representation (wrap with ST\_AsEWKT) -

SRID=4326;POINT(-123.365556 48.428611)

-- Mark a point as WGS 84 long lat and then transform to web mercator (Spherical Mercator) --

SELECT ST\_Transform(ST\_SetSRID(ST\_Point(-123.365556, 48.428611),4326),3785) As spere\_merc;

-- the ewkt representation (wrap with ST\_AsEWKT) -

SRID=3785;POINT(-13732990.8753491 6178458.96425423)

**请参考**

Section 4.3.1, ST\_AsEWKT, ST\_Point, ST\_SRID, ST\_Transform, UpdateGeometrySRID

**8.6.26**

**ST\_SnapToGrid**

ST\_SnapToGrid — 把输入的几何对象的所有点重新组成一个规则的栅格

**用法**

geometry ST\_SnapToGrid(geometry geomA, ﬂoat originX, ﬂoat originY, ﬂoat sizeX, ﬂoat sizeY);

geometry ST\_SnapToGrid(geometry geomA, ﬂoat sizeX, ﬂoat sizeY);

geometry ST\_SnapToGrid(geometry geomA, ﬂoat size);

geometry ST\_SnapToGrid(geometry geomA, geometry pointOrigin, ﬂoat sizeX, ﬂoat sizeY, ﬂoat sizeZ, ﬂoat sizeM);



PostGIS 2.2.0dev 开发手册

189 / 686

**描述**

函数形式 1,2,3: 把输入的几何对象的所有点重新组成一个以该对象中心点和它的像元大小的栅格。该函数会删除同一个像元上的连续的点，如果该函数输出的点不足以生成给定的几何类型的对象，那么返回值就是NULL。该函数会把输入的几何对象内部折叠的几何对象删除掉。这个函数对降低栅格分辨率很有用。函数形式 4: 1.1.0版本引入， - 把输入的几何对象的所有点重新组成一个以该对象中心点和它的像元大小的栅格 (第二个参数必须是point类型对象).如果你不想把输入对象转变成栅格，把所有维度值都设置为0

**注意**

该函数返回的geometry对象可能会失去简单性(参考 ST\_IsSimple).

**注意**

在1.1.0版本之前，该总是会返回一个2D几何对象。从1.1.0版本开始，该函数会返回一个和输入对象更高的相同的维度值。使用带有第二个几何对象参数的函数（译者注：即第四种形式）来定义所有网格的维度。

可用版本: 1.0.0RC1

可用版本: 1.1.0 - Z and M support

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

**样例**

--Snap your geometries to a precision grid of 10^-3

UPDATE mytable

SET the\_geom = ST\_SnapToGrid(the\_geom, 0.001);

SELECT ST\_AsText(ST\_SnapToGrid(

ST\_GeomFromText(’LINESTRING(1.1115678 2.123, 4.111111 3.2374897, 4.11112 3.23748667) ←

’),

0.001)

);

st\_astext

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

LINESTRING(1.112 2.123,4.111 3.237)

--Snap a 4d geometry

SELECT ST\_AsEWKT(ST\_SnapToGrid(

ST\_GeomFromEWKT(’LINESTRING(-1.1115678 2.123 2.3456 1.11111,

4.111111 3.2374897 3.1234 1.1111, -1.11111112 2.123 2.3456 1.1111112)’),

ST\_GeomFromEWKT(’POINT(1.12 2.22 3.2 4.4444)’),

0.1, 0.1, 0.1, 0.01) );

st\_asewkt

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

LINESTRING(-1.08 2.12 2.3 1.1144,4.12 3.22 3.1 1.1144,-1.08 2.12 2.3 1.1144)

--With a 4d geometry - the ST\_SnapToGrid(geom,size) only touches x and y coords but keeps m ←

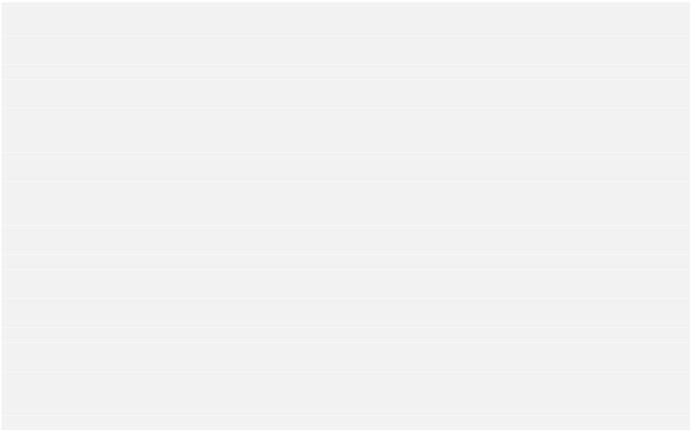
and z the same

SELECT ST\_AsEWKT(ST\_SnapToGrid(ST\_GeomFromEWKT(’LINESTRING(-1.1115678 2.123 3 2.3456,

4.111111 3.2374897 3.1234 1.1111)’),

0.01) );

st\_asewkt



PostGIS 2.2.0dev 开发手册

190 / 686

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

LINESTRING(-1.11 2.12 3 2.3456,4.11 3.24 3.1234 1.1111)

**请参考**

ST\_Snap, ST\_AsEWKT, ST\_AsText, ST\_GeomFromText, ST\_GeomFromEWKT, ST\_Simplify

**8.6.27**

**ST\_Snap**

ST\_Snap — 把一个输入的几何对象的片段和顶点切割后重新组装成一个参考类型的几何对象

**用法**

geometry ST\_Snap(geometry input, geometry reference, ﬂoat tolerance);

**描述**

把一个输入的几何对象的片段和顶点切割后移向一个参考类型的几何对象的顶点上（译者注：直观上看就是把第一个几何对象移向第二个几何对象，让两者靠近），.一个最大的切割距离用来控制在片段的哪个位置进行切割。切割一个几何对象并组装成另一个可以提高消除相近边界（消除边界时在节点交集计算时会引起问题）带来的覆盖操作的鲁棒性。太多次的切割会引起生成的拓扑结构无效。因此切割点的位置和切割次数是使用探索法来决定怎样进行安全的切割。然而这会导致有一些切割被省略掉。

**注意**

返回的几何对象可能会丢失简单性 (参考 ST\_IsSimple) 和有效性(参考 ST\_IsValid).

可用版本: 2.0.0需要GEOS版本>= 3.3.0.

**样例**



PostGIS 2.2.0dev 开发手册

191 / 686

一个没有任何切割的用LINESTRING展示的multipolygon 对象



PostGIS 2.2.0dev 开发手册

192 / 686

A multipolygon snapped to linestring to tolerance: 1.25 of

A multipolygon snapped to linestring to tolerance: 1.01 of

distance. The new multipolygon is shown with reference

distance. The new multipolygon is shown with reference

linestring

linestring

SELECT ST\_AsText(

SELECT ST\_AsText(ST\_Snap(poly,line,

←

ST\_Snap(poly,line, ST\_Distance(poly, ←

ST\_Distance(poly,line)\*1.01)) AS polysnapped

line)\*1.25)

FROM (SELECT

ST\_GeomFromText(’MULTIPOLYGON(

) AS polysnapped

FROM (SELECT

((26 125, 26 200, 126 200, 126 125,

←

ST\_GeomFromText(’MULTIPOLYGON(

26 125 ),

(( 26 125, 26 200, 126 200, 126 125,

←

( 51 150, 101 150, 76 175, 51 150 ) ←

),

26 125 ),

( 51 150, 101 150, 76 175, 51 150 ) ←

(( 151 100, 151 200, 176 175, 151

←

),

100 )))’) As poly,

(( 151 100, 151 200, 176 175, 151

←

ST\_GeomFromText(’LINESTRING (5

←

100 )))’) As poly,

107, 54 84, 101 100)’) As line

) As foo;

ST\_GeomFromText(’LINESTRING (5

107, 54 84, 101 100)’) As line

←

) As foo;

polysnapped

--------------------------------------------------------------------- ←

polysnapped

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

MULTIPOLYGON(((26 125,26 200,126 200,126 ←

125,101 100,26 125),

MULTIPOLYGON(((5 107,26 200,126 200,126

←

(51 150,101 150,76 175,51 150)),((151

←

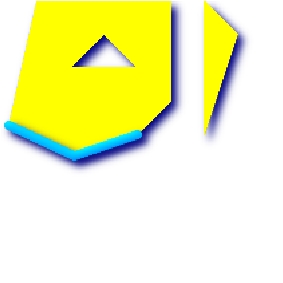
125,101 100,54 84,5 107),

100,151 200,176 175,151 100)))

(51 150,101 150,76 175,51 150)),((151

100,151 200,176 175,151 100)))

←



PostGIS 2.2.0dev 开发手册

193 / 686

The linestring snapped to the original multipolygon at

tolerance 1.01 of distance. The new linestring is shown

with reference multipolygon

SELECT ST\_AsText(

ST\_Snap(line, poly, ST\_Distance(poly, ←

line)\*1.01)

) AS linesnapped

FROM (SELECT

ST\_GeomFromText(’MULTIPOLYGON(

((26 125, 26 200, 126 200, 126 125, ←

26 125),

(51 150, 101 150, 76 175, 51 150 )) ←

,

((151 100, 151 200, 176 175, 151 ←

100)))’) As poly,

ST\_GeomFromText(’LINESTRING (5 ←

107, 54 84, 101 100)’) As line

) As foo;

linesnapped

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

LINESTRING(5 107,26 125,54 84,101 100)

**请参考**

ST\_SnapToGrid

The linestring snapped to the original multipolygon at

tolerance 1.25 of distance. The new linestring is shown

with reference multipolygon

SELECT ST\_AsText(

ST\_Snap(line, poly, ST\_Distance(poly, ←

line)\*1.25)

) AS linesnapped

FROM (SELECT

ST\_GeomFromText(’MULTIPOLYGON(

(( 26 125, 26 200, 126 200, 126 125, ←

26 125 ),

(51 150, 101 150, 76 175, 51 150 )) ←

,

((151 100, 151 200, 176 175, 151 ←

100 )))’) As poly,

ST\_GeomFromText(’LINESTRING (5 ←

107, 54 84, 101 100)’) As line

) As foo;

linesnapped

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

LINESTRING(26 125,54 84,101 100)

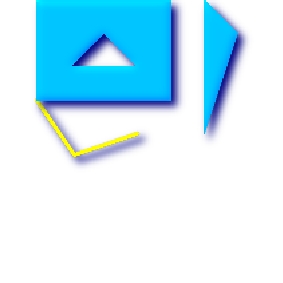
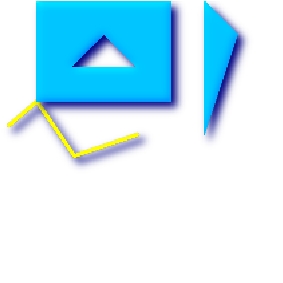
**8.6.28**

**ST\_Transform**

ST\_Transform — 返回一个根据给定的SRID转换成新的坐标系对象

**用法**

geometry ST\_Transform(geometry g1, integer srid);



PostGIS 2.2.0dev 开发手册

194 / 686

**描述**

返回一个根据给定的SRID转换成新的坐标系对象，SRID值必须存在于 SPATIAL\_REF\_SYS 表中。ST\_Transform 函数通常会和函数ST\_SetSRID()混淆. ST\_Transform函数实际上会把一个几何对象坐标从一个参考系转换成另一个，而函数ST\_SetSRID() 只是简单的修改输入的几何对象的SRID值

**注意**

该函数要使用的话，PostGIS必须在安装时候添加库Proj一起编译安装. 使用函数PostGIS\_Full\_Version来确认你已经添加了proj库支持

**注意**

如果你多次使用同一个转换，在经常使用该函数的列上添加该函数索引是很有用的，这样可以充分利用索引

**注意**

1.3.4版本前,该函数如果包含CURVE类型对象会crash掉，1.3.4+版本均已修复

版本提升: 2.0.0 版本开始支持Polyhedral Surface几何对象.

该函数方法实现了规范 SQL/MM speciﬁcation. SQL-MM 3: 5.1.6

该函数支持CircularString和Curve几何类型对象

该函数支持 Polyhedral Surface类型几何对象.

**样例**

Change Mass state plane US feet geometry to WGS 84 long lat

SELECT ST\_AsText(ST\_Transform(ST\_GeomFromText(’POLYGON((743238 2967416,743238 2967450,

743265 2967450,743265.625 2967416,743238 2967416))’,2249),4326)) As wgs\_geom;

wgs\_geom

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

POLYGON((-71.1776848522251 42.3902896512902,-71.1776843766326 42.3903829478009,

-71.1775844305465 42.3903826677917,-71.1775825927231 42.3902893647987,-71.177684

8522251 42.3902896512902));

(1 row)

--3D Circular String example

SELECT ST\_AsEWKT(ST\_Transform(ST\_GeomFromEWKT(’SRID=2249;CIRCULARSTRING(743238 2967416

1,743238 2967450 2,743265 2967450 3,743265.625 2967416 3,743238 2967416 4)’),4326));

st\_asewkt

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

←

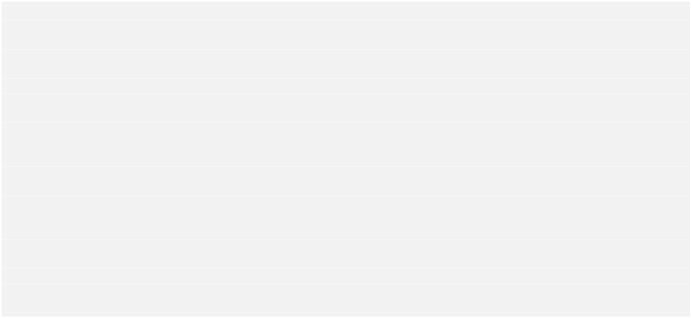
SRID=4326;CIRCULARSTRING(-71.1776848522251 42.3902896512902 1,-71.1776843766326

42.3903829478009 2,

-71.1775844305465 42.3903826677917 3,

-71.1775825927231 42.3902893647987 3,-71.1776848522251 42.3902896512902 4)

←



PostGIS 2.2.0dev 开发手册

195 / 686

创建函数索引的例子.对于那些你不确定将会记录那些几何对象的表，最好的办法是使用partial index（译者注：即部分索引，该索引是PostgreSQL的一种索引类型）类型的索引，该索引会忽略掉NULL类型几何对象，但会保留值为空即’’的值，这会让你的索引更小更有效率。

CREATE INDEX idx\_the\_geom\_26986\_parcels

ON parcels

USING gist

(ST\_Transform(the\_geom, 26986))

WHERE the\_geom IS NOT NULL;

**配置转换方式**

有时候坐标转换时候，涉及到栅格位移时候会导致失败，例如如果PROJ.4 安装时候没有栅格位移文件或者坐标不处在栅格定义的位移范围内。默认情况下，PostGIS在没有提供栅格位移文件时候会抛出一个错误，但是这种方式可以通过改变在spatial\_ref\_sys table表中的proj4text值来配置每一个SRID基底。例如proj4text参数+datum=NAD87是如下+nadgrids参数的简单等效办法：

+nadgrids=@conus,@alaska,@ntv2\_0.gsb,@ntv1\_can.dat

参数 @ preﬁx 意义是如果没有提供栅格位移文件，将不会报错，但是如果搜索到文件列表结束后，没有找到合适的文件，将会报错。

相反，如果你保证最基本的文件已经提供了，但扫描所有文件后了却没有找到对应的转换，你应该使用下面的参数

+nadgrids=@conus,@alaska,@ntv2\_0.gsb,@ntv1\_can.dat,null

参数null 栅格位移文件是一个有效的能够覆盖所有地方的栅格位移文件，作用是保证没有位移，因此作为一个完整的例子，如果你想修改一个SRID=4267不在正常范围内的转换，并且保证不报ERROR，你应该使用如下的方式：

UPDATE spatial\_ref\_sys SET proj4text = ’+proj=longlat +ellps=clrk66 +nadgrids=@conus, ←

@alaska,@ntv2\_0.gsb,@ntv1\_can.dat,null +no\_defs’ WHERE srid = 4267;

**请参考**

PostGIS\_Full\_Version, ST\_AsText, ST\_SetSRID, UpdateGeometrySRID

**8.6.29**

**ST\_Translate**

ST\_Translate — 把一个几何对象根据指定的输入偏移地址，移动到一个新的位置。例如ST\_Translate(geom, X,Y) 或者 ST\_Translate(geom, X, Y,Z).

**用法**

geometry ST\_Translate(geometry g1, ﬂoat deltax, ﬂoat deltay);

geometry ST\_Translate(geometry g1, ﬂoat deltax, ﬂoat deltay, ﬂoat deltaz);

**描述**

把一个几何对象根据指定的输入偏移地址，移动到一个新的位置。移动便宜地址为delta x,delta y,delta z单位，单位的定义由SRID值决定

PostGIS 2.2.0dev 开发手册

196 / 686

**注意**

1.3.4版本前,该函数如果包含CURVE类型对象会crash掉，1.3.4+版本均已修复

可用版本: 1.2.2

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

该函数支持CircularString和Curve几何类型对象

**样例**

Move a point 1 degree longitude

SELECT ST\_AsText(ST\_Translate(ST\_GeomFromText(’POINT(-71.01 42.37)’,4326),1,0)) As

←

wgs\_transgeomtxt;

wgs\_transgeomtxt

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

POINT(-70.01 42.37)

Move a linestring 1 degree longitude and 1/2 degree latitude

SELECT ST\_AsText(ST\_Translate(ST\_GeomFromText(’LINESTRING(-71.01 42.37,-71.11 42.38)’,4326) ←

,1,0.5)) As wgs\_transgeomtxt;

wgs\_transgeomtxt

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

LINESTRING(-70.01 42.87,-70.11 42.88)

Move a 3d point

SELECT ST\_AsEWKT(ST\_Translate(CAST(’POINT(0 0 0)’ As geometry), 5, 12,3));

st\_asewkt

---------

POINT(5 12 3)

Move a curve and a point

SELECT ST\_AsText(ST\_Translate(ST\_Collect(’CURVEPOLYGON(CIRCULARSTRING(4 3,3.12 0.878,1

0,-1.121 5.1213,6 7, 8 9,4 3))’,’POINT(1 3)’),1,2));

st\_astext

←

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

GEOMETRYCOLLECTION(CURVEPOLYGON(CIRCULARSTRING(5 5,4.12 2.878,2 2,-0.121 7.1213,7 9,9 11,5 ←

5)),POINT(2 5))

**请参考**

ST\_Afﬁne, ST\_AsText, ST\_GeomFromText

**8.6.30**

**ST\_TransScale**

ST\_TransScale — 根据输入的缩放倍数，对一个几何对象进行缩放，缩放因子为Xfactor和Yfactor参数，该函数只支持2D对象



PostGIS 2.2.0dev 开发手册

197 / 686

**用法**

geometry ST\_TransScale(geometry geomA, ﬂoat deltaX, ﬂoat deltaY, ﬂoat XFactor, ﬂoat YFactor);

**描述**

根据输入的缩放倍数，对一个几何对象进行缩放，缩放因子为Xfactor和Yfactor参数，该函数只支持2D对象

**注意**

ST\_TransScale(geomA, deltaX, deltaY, XFactor, YFactor)是函数ST\_Affine(geomA, XFactor, 0, 0, 0, YFactor, 0, 0, 0, 1, deltaX\*XFactor, deltaY\*YFactor,0)的简化版（译者注：仿射函数ST\_Affine参数太多了，这个函数是仿射函数的一种特殊情况的简化版，相比较而言更简单一些）

**注意**

1.3.4版本前,该函数如果包含CURVE类型对象会crash掉，1.3.4+版本均已修复

可用版本: 1.1.0.

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

该函数支持CircularString和Curve几何类型对象

**样例**

SELECT ST\_AsEWKT(ST\_TransScale(ST\_GeomFromEWKT(’LINESTRING(1 2 3, 1 1 1)’), 0.5, 1, 1, 2));

st\_asewkt

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

LINESTRING(1.5 6 3,1.5 4 1)

--Buffer a point to get an approximation of a circle, convert to curve and then translate

1,2 and scale it 3,4

SELECT ST\_AsText(ST\_Transscale(ST\_LineToCurve(ST\_Buffer(’POINT(234 567)’, 3)),1,2,3,4));

st\_astext

←

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

CURVEPOLYGON(CIRCULARSTRING(714 2276,711.363961030679 2267.51471862576,705

2264,698.636038969321 2284.48528137424,714 2276))

**请参考**

ST\_Afﬁne, ST\_Translate

←

**8.7**

**8.7.1**

**Geometry Outputs**

**ST\_AsBinary**

ST\_AsBinary — 返回一个没有SRID信息的WKB描述的geometry或geography对象.



PostGIS 2.2.0dev 开发手册

198 / 686

**用法**

bytea ST\_AsBinary(geometry g1);

bytea ST\_AsBinary(geometry g1, text NDR\_or\_XDR);

bytea ST\_AsBinary(geography g1);

bytea ST\_AsBinary(geography g1, text NDR\_or\_XDR);

**描述**

返回一个WKB描述的几何对象。这个函数有两种变体，第一种是没有输入编码参数的，而是使用服务端的机器编码方式。第二种有一个编码参数，对’NDR’数据使用little-endian编码，对XDR数据使用big-endian (’XDR’) 编码

在使用二进制游标（译者注：二进制游标是PostgreSQL支持的一种游标）将导出数据库时候，不用讲二进制数据转化成文本形式，这一点很有用

**注意**

WKB SPEC规范不包含关于SRID的要求，如果想得到带有SRID描述的WKB对象，请使用函数ST\_AsEWKB

**注意**

对几何对象来说，函数ST\_AsBinary是函数ST\_GeomFromWKB逆功能函数，使用函数ST\_GeomFromWKB可以把一个ST\_AsBinary 表述的几何对象转化成一个postgis的几何对象

**注意**

在PostgreSQL 9.0数据库中，该函数的默认输出bytea对象是用八进制编码的。对一个几何对象来说，函数ST\_AsBinary是ST\_GeomFromWKB的逆功能函数. 如果你的GUI工具需要老的方式，那么需要在你的数据库连接session中做如下设置SET bytea\_output=’escape’

版本提升: 2.0.0 版本支持Polyhedral Surface类型、Triangle类型、TIN类型.

版本提升: 2.0.0引入支持更高维度坐标.

版本提升: 2.0.0 支持指定geography编码

可用版本: 1.5.0 支持geography对象.

版本变更: 2.0.0 该函数的输入不能是未知的，必须是一个几何对象。如下查询SELECT ST\_AsBinary(’POINT(1 2)’) 不再支持，你会得到一个st\_asbinary(unknown) 不唯一的错误.应该把代码修改为ST\_AsBinary(’POINT(1 2)’::geometry);. 如果该查询不可永，那么需要执行代码文件legacy.sql.（译者注：使用psql执行）

该函数方法实现了规范 OpenGIS Simple Features Implementation Speciﬁcation for SQL 1.1. s2.1.1.1

该函数方法实现了规范 SQL/MM speciﬁcation. SQL-MM 3: 5.1.37

该函数支持CircularString和Curve几何类型对象

该函数支持 Polyhedral Surface类型几何对象.

该函数支持 Triangles 和 Triangulated Irregular Network Surfaces (TIN)类型.

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.



PostGIS 2.2.0dev 开发手册

199 / 686

**样例**

SELECT ST\_AsBinary(ST\_GeomFromText(’POLYGON((0 0,0 1,1 1,1 0,0 0))’,4326));

st\_asbinary

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

\001\003\000\000\000\001\000\000\000\005

\000\000\000\000\000\000\000\000\000\000

\000\000\000\000\000\000\000\000\000\000

\000\000\000\000\000\000\000\000\000\000

\000\000\000\360?\000\000\000\000\000\000

\360?\000\000\000\000\000\000\360?\000\000

\000\000\000\000\360?\000\000\000\000\000

\000\000\000\000\000\000\000\000\000\000\000

\000\000\000\000\000\000\000\000

(1 row)

SELECT ST\_AsBinary(ST\_GeomFromText(’POLYGON((0 0,0 1,1 1,1 0,0 0))’,4326), ’XDR’);

st\_asbinary

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

\000\000\000\000\003\000\000\000\001\000\000\000\005\000\000\000\000\000

\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000

\000?\360\000\000\000\000\000\000?\360\000\000\000\000\000\000?\360\000\000

\000\000\000\000?\360\000\000\000\000\000\000\000\000\000\000\000\000\000\000

\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000

(1 row)

**请参考**

ST\_GeomFromWKB ST\_AsEWKB, ST\_AsText,

**8.7.2**

**ST\_AsEWKB**

ST\_AsEWKB —返回一个带有SRID信息的WKB描述的geometry对象.

**用法**

bytea ST\_AsEWKB(geometry g1);

bytea ST\_AsEWKB(geometry g1, text NDR\_or\_XDR);

**描述**

返回一个WKB描述的几何对象。这个函数有两种变体，第一种是没有输入编码参数的，而是使用服务端的机器编码方式。第二种有一个编码参数，对’NDR’数据使用little-endian编码，对XDR数据使用big-endian (’XDR’) 编码

在使用二进制游标（译者注：二进制游标是PostgreSQL支持的一种游标）将导出数据库时候，不用讲二进制数据转化成文本形式，这一点很有用

**注意**

WKB SPEC规范不包含关于SRID的要求，如果想得到带有SRID描述的OGC WKB对象，请使用函数ST\_AsBinary



PostGIS 2.2.0dev 开发手册

200 / 686

**注意**

函数ST\_AsEWKB是函数ST\_ GeomFromEWKB逆功能函数，使用函数ST\_GeomFromEWKB 可以把一个ST\_AsEWKB表述的几何对象转化成一个postgis的几何对象

版本提升: 2.0.0 版本支持Polyhedral Surface类型、Triangle类型、TIN类型.

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

该函数支持CircularString和Curve几何类型对象

该函数支持 Polyhedral Surface类型几何对象.

该函数支持 Triangles 和 Triangulated Irregular Network Surfaces (TIN)类型.

**样例**

SELECT ST\_AsEWKB(ST\_GeomFromText(’POLYGON((0 0,0 1,1 1,1 0,0 0))’,4326));

st\_asewkb

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

\001\003\000\000 \346\020\000\000\001\000

\000\000\005\000\000\000\000

\000\000\000\000\000\000\000\000

\000\000\000\000\000\000\000\000\000

\000\000\000\000\000\000\000\000\000\000

\000\000\360?\000\000\000\000\000\000\360?

\000\000\000\000\000\000\360?\000\000\000\000\000

\000\360?\000\000\000\000\000\000\000\000\000\000\000

\000\000\000\000\000\000\000\000\000\000\000\000\000

(1 row)

SELECT ST\_AsEWKB(ST\_GeomFromText(’POLYGON((0 0,0 1,1 1,1 0,0 0))’,4326), ’XDR’);

st\_asewkb

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

\000 \000\000\003\000\000\020\346\000\000\000\001\000\000\000\005\000\000\000\000\

000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000?

\360\000\000\000\000\000\000?\360\000\000\000\000\000\000?\360\000\000\000\000

\000\000?\360\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000

\000\000\000\000\000\000\000\000\000\000\000\000\000

**请参考**

ST\_AsBinary, ST\_AsEWKT, ST\_AsText, ST\_GeomFromEWKT, ST\_SRID

**8.7.3**

**ST\_AsEWKT**

ST\_AsEWKT —返回一个带有SRID信息的WKT描述的geometry对象.

**用法**

text ST\_AsEWKT(geometry g1);

text ST\_AsEWKT(geography g1);



PostGIS 2.2.0dev 开发手册

201 / 686

**描述**

返回一个带有SRID信息的WKT描述的geometry对象.

**注意**

WKT SPEC规范不包含SRID.想得到OGC WKT 格式的，使用函数ST\_AsText

WKT 格式不能保证坐标精度，因此为了阻止精度损失，使用函数 ST\_AsBinary 或函数 ST\_AsEWKB .

**注意**

ST\_AsEWKT is the reverse of ST\_GeomFromEWKT. Use ST\_GeomFromEWKT to convert to a postgis geometry from

ST\_AsEWKT representation.

版本提升: 2.0.0 support for Geography, Polyhedral surfaces, Triangles and TIN was introduced.

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

该函数支持CircularString和Curve几何类型对象

该函数支持 Polyhedral Surface类型几何对象.

该函数支持 Triangles 和 Triangulated Irregular Network Surfaces (TIN)类型.

**样例**

SELECT ST\_AsEWKT(’0103000020E61000000100000005000000000000

000000000000000000000000000000000000000000000000000000

F03F000000000000F03F000000000000F03F000000000000F03

F000000000000000000000000000000000000000000000000’::geometry);

st\_asewkt

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

SRID=4326;POLYGON((0 0,0 1,1 1,1 0,0 0))

(1 row)

SELECT ST\_AsEWKT(’0108000080030000000000000060 ←

E30A4100000000785C0241000000000000F03F0000000018

E20A4100000000485F024100000000000000400000000018

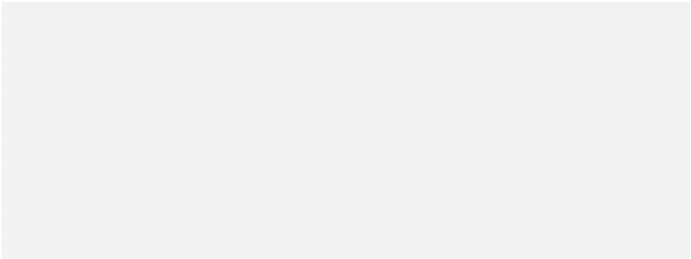
E20A4100000000305C02410000000000000840’)

--st\_asewkt---

CIRCULARSTRING(220268 150415 1,220227 150505 2,220227 150406 3)

**请参考**

ST\_AsBinaryST\_AsEWKBST\_AsText, ST\_GeomFromEWKT



PostGIS 2.2.0dev 开发手册

202 / 686

**8.7.4**

**ST\_AsGeoJSON**

ST\_AsGeoJSON — 返回一个GeoJSON格式（译者注：GeoJSON是一种类似json的格式数据，适用于地理编码）的几何对象.

**用法**

text ST\_AsGeoJSON(geometry geom, integer maxdecimaldigits=15, integer options=0);

text ST\_AsGeoJSON(geography geog, integer maxdecimaldigits=15, integer options=0);

text ST\_AsGeoJSON(integer gj\_version, geometry geom, integer maxdecimaldigits=15, integer options=0);

text ST\_AsGeoJSON(integer gj\_version, geography geog, integer maxdecimaldigits=15, integer options=0);

**描述**

返回一个GeoJSON格式的几何对象. (参考 GeoJSON speciﬁcations 1.0). 2D 和3D 几何对象都支持. GeoJSON 只支持SFS 1.1 规范的几何类型 (例如不支持curve 对象类型).

gj\_version 参数是GeoJSON SPEC规范的主版本号（译者注：例如GeoJSON规范1.0，则主版本号是1，子版本号是0），如果指定了该值，该值必须设置我1.这表示GeoJSON SPEC的主版本号。第三个参数可能用在减少输出精度（默认小数位是15位）的场景上。最后一个参数”option”可来控制在GeoJSON输出中添加Bbox或Crs标签

· 0: 没有option选项，是默认情况

· 1: 添加GeoJSON Bbox标签元素

· 2:输出添加 GeoJSON Short CRS (e.g EPSG:4326)标签元素

· 4: 输出添加GeoJSON Long CRS (e.g urn:ogc:def:crs:EPSG::4326)标签元素

形式 1: ST\_AsGeoJSON(geom) / precision=15 version=1 options=0

形式2: ST\_AsGeoJSON(geom, precision) / version=1 options=0

形式3: ST\_AsGeoJSON(geom, precision, options) / version=1

形式4: ST\_AsGeoJSON(gj\_version, geom) / precision=15 options=0

形式5: ST\_AsGeoJSON(gj\_version, geom, precision) /options=0

形式6: ST\_AsGeoJSON(gj\_version, geom, precision,options)

可用版本: 1.3.4

可用版本: 1.5.0 引入支持geography对象.

版本变更: 2.0.0 support default args and named args.

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

**样例**

在ajax通信中，GeoJSON 格式通常是比其他格式文档更高效的格式。一个流行的JavaScript 客户端Open Layers支持这种数据格式。关于该数据格式的使用样例参考 [OpenLayers GeoJSON Example](http://openlayers.org/dev/examples/vector-formats.html)

SELECT ST\_AsGeoJSON(the\_geom) from fe\_edges limit 1;

st\_asgeojson

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

{"type":"MultiLineString","coordinates":[[[-89.734634999999997,31.492072000000000],



PostGIS 2.2.0dev 开发手册

203 / 686

[-89.734955999999997,31.492237999999997]]]}

(1 row)

--3d point

SELECT ST\_AsGeoJSON(’LINESTRING(1 2 3, 4 5 6)’);

st\_asgeojson

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

{"type":"LineString","coordinates":[[1,2,3],[4,5,6]]}

**8.7.5**

**ST\_AsGML**

ST\_AsGML —返回GML 2或GML 3版本格式描述的几何对象

**用法**

text ST\_AsGML(integer version, geometry geom, integer maxdecimaldigits=15, integer options=0, text npreﬁx=null, text id=null);

text ST\_AsGML(integer version, geography geog, integer maxdecimaldigits=15, integer options=0, text npreﬁx=null, text

id=null);

**描述**

返回地理标记语言（GML）格式输出，如果指定版本参数的话，该版本参数只能是2或者3，如果没有指定版本号，默认值为2。精度参数可以用来减小输出的精度，默认是15位小数

在这里GML 2 使用的是 2.1.2 版本, GML 3 使用 3.1.1 版本

参数options是一个标志位，可以用来在GML输出中定义CRS输出类型，以及以经纬度方式定义数据

· 0: GML Short CRS (e.g EPSG:4326),默认值

· 1: GML Long CRS (e.g urn:ogc:def:crs:EPSG::4326)

· 2: 只支持GML 3 版本的参数,从输出中移除srsDimension 标签属性.

· 4: 只支持GML 3 版本的参数, 对于LINESTRING对象使用<LineString>而不是<Curve>标签.

· 16: 声明数据是经纬度格式(例如 srid=4326). 默认假定的输出的是二维平面数据。该参数对GML 3.1.1版本输出很有用，和轴顺序相关。因此如果你设定了该值，该函数会交换坐标顺序为lat/lon的顺序，而不是lon/lat的顺序

· 32:输出geometry的包络

参数’namespace preﬁx’ 可以用来指定一个定制的gml文档的namespace preﬁx，如果没有指定则没有prefix，如果为null或者省略该参数，则prefix为‘gml’

可用版本: 1.3.2

可用版本: 1.5.0 引入支持geography对象.

版本提升:2.0.0引入对prefix的支持，引入了对GML3版本的参数option为4的支持，以便允许line对象使用LineString而不是Curve这样的标签。引入了对于Polyhedral surfaces and TINS 对象的GML3 的支持。引入了对于参数为option为32时候的支持。

版本变更: 2.0.0 使用默认的命名参数

版本提升: 2.1.0 引入对于GML3版本的id支持

PostGIS 2.2.0dev 开发手册

204 / 686

**注意**

只有GML 3+ 版本的函数ST\_AsGML才支持Polyhedral Surface和TIN类型对象.

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

该函数支持 Polyhedral Surface类型几何对象.

该函数支持 Triangles 和 Triangulated Irregular Network Surfaces (TIN)类型.

**样例: Version 2**

SELECT ST\_AsGML(ST\_GeomFromText(’POLYGON((0 0,0 1,1 1,1 0,0 0))’,4326));

st\_asgml

--------

<gml:Polygon srsName="EPSG:4326"><gml:outerBoundaryIs><gml:LinearRing><gml:coordinates ←

>0,0 0,1 1,1 1,0 0,0</gml:coordinates></gml:LinearRing></gml:outerBoundaryIs></gml: ←

Polygon>

**样例: Version 3**

-- Flip coordinates and output extended EPSG (16 | 1)--

SELECT ST\_AsGML(3, ST\_GeomFromText(’POINT(5.234234233242 6.34534534534)’,4326), 5, 17);

st\_asgml

--------

<gml:Point srsName="urn:ogc:def:crs:EPSG::4326"><gml:pos>6.34535 5.23423</gml:pos></gml ←

:Point>

-- Output the envelope (32) --

SELECT ST\_AsGML(3, ST\_GeomFromText(’LINESTRING(1 2, 3 4, 10 20)’,4326), 5, 32);

st\_asgml

--------

<gml:Envelope srsName="EPSG:4326">

<gml:lowerCorner>1 2</gml:lowerCorner>

<gml:upperCorner>10 20</gml:upperCorner>

</gml:Envelope>

-- Output the envelope (32) , reverse (lat lon instead of lon lat) (16), long srs (1)= 32 | ←

16 | 1 = 49 --

SELECT ST\_AsGML(3, ST\_GeomFromText(’LINESTRING(1 2, 3 4, 10 20)’,4326), 5, 49);

st\_asgml

--------

<gml:Envelope srsName="urn:ogc:def:crs:EPSG::4326">

<gml:lowerCorner>2 1</gml:lowerCorner>

<gml:upperCorner>20 10</gml:upperCorner>

</gml:Envelope>

-- Polyhedral Example --

SELECT ST\_AsGML(3, ST\_GeomFromEWKT(’POLYHEDRALSURFACE( ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0) ←

),

((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),

((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),

((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)) )’));



PostGIS 2.2.0dev 开发手册

205 / 686

st\_asgml

--------

<gml:PolyhedralSurface>

<gml:polygonPatches>

<gml:PolygonPatch>

<gml:exterior>

<gml:LinearRing>

<gml:posList srsDimension="3">0

</gml:LinearRing>

</gml:exterior>

</gml:PolygonPatch>

<gml:PolygonPatch>

<gml:exterior>

<gml:LinearRing>

<gml:posList srsDimension="3">0

</gml:LinearRing>

</gml:exterior>

</gml:PolygonPatch>

<gml:PolygonPatch>

<gml:exterior>

<gml:LinearRing>

<gml:posList srsDimension="3">0

</gml:LinearRing>

</gml:exterior>

</gml:PolygonPatch>

<gml:PolygonPatch>

<gml:exterior>

<gml:LinearRing>

<gml:posList srsDimension="3">1

</gml:LinearRing>

</gml:exterior>

</gml:PolygonPatch>

<gml:PolygonPatch>

<gml:exterior>

<gml:LinearRing>

<gml:posList srsDimension="3">0

</gml:LinearRing>

</gml:exterior>

</gml:PolygonPatch>

<gml:PolygonPatch>

<gml:exterior>

<gml:LinearRing>

<gml:posList srsDimension="3">0

</gml:LinearRing>

</gml:exterior>

</gml:PolygonPatch>

</gml:polygonPatches>

</gml:PolyhedralSurface>

**请参考**

ST\_GeomFromGML

0 0 0 0 1 0 1 1 0 1 0 0 0 0</gml:posList>

0 0 0 1 0 1 1 0 1 0 0 0 0 0</gml:posList>

0 0 1 0 0 1 0 1 0 0 1 0 0 0</gml:posList>

1 0 1 1 1 1 0 1 1 0 0 1 1 0</gml:posList>

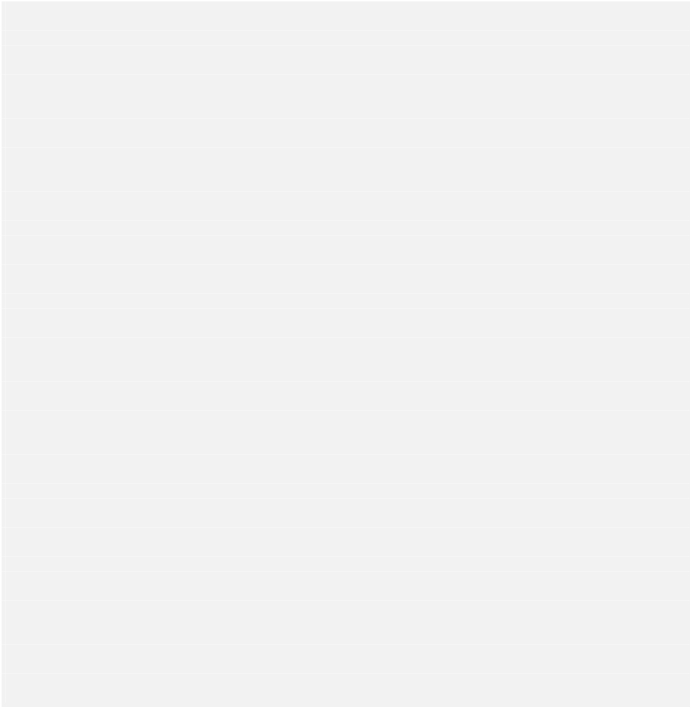
1 0 0 1 1 1 1 1 1 1 0 0 1 0</gml:posList>

0 1 1 0 1 1 1 1 0 1 1 0 0 1</gml:posList>

**8.7.6**

**ST\_AsHEXEWKB**

ST\_AsHEXEWKB — 返回HEXEWKB格式（文本表示）的geometry对象，对NDR数据使用little-endian编码，对XDR数据使用big-endian编码



PostGIS 2.2.0dev 开发手册

206 / 686

**用法**

text ST\_AsHEXEWKB(geometry g1, text NDRorXDR);

text ST\_AsHEXEWKB(geometry g1);

**描述**

返回一个用HEXEWKB表示的几何对象，对NDR数据使用little-endian编码，对XDR数据使用big-endian编码。如果没有指定编码方式，默认使用NDR数据格式

.

**注意**

可用版本: 1.2.2

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

该函数支持CircularString和Curve几何类型对象

**样例**

SELECT ST\_AsHEXEWKB(ST\_GeomFromText(’POLYGON((0 0,0 1,1 1,1 0,0 0))’,4326));

which gives same answer as

SELECT ST\_GeomFromText(’POLYGON((0 0,0 1,1 1,1 0,0 0))’,4326)::text;

st\_ashexewkb

--------

0103000020E6100000010000000500

00000000000000000000000000000000

00000000000000000000000000000000F03F

000000000000F03F000000000000F03F000000000000F03

F000000000000000000000000000000000000000000000000

**8.7.7**

**ST\_AsKML**

ST\_AsKML — 返回一个几何对象的KML 元素表示。该函数有几种形式，该函数默认的version值为2，默认的precision值为15

**用法**

text ST\_AsKML(geometry geom, integer maxdecimaldigits=15);

text ST\_AsKML(geography geog, integer maxdecimaldigits=15);

text ST\_AsKML(integer version, geometry geom, integer maxdecimaldigits=15, text npreﬁx=NULL);

text ST\_AsKML(integer version, geography geog, integer maxdecimaldigits=15, text npreﬁx=NULL);

**描述**

返回一个几何对象的KML 元素表示。该函数有几种形式，该函数默认的version值为2，默认的precision值为15。该函数有几种形式。输出的坐标最大精度默认为15位小数，version默认值为2，默认的KML输出namespace没有前缀

形式 1: ST\_AsKML(geom\_or\_geog, maxdecimaldigits) / version=2 / maxdecimaldigits=15

形式 2: ST\_AsKML(version, geom\_or\_geog, maxdecimaldigits, npreﬁx) maxdecimaldigits=15 / npreﬁx=NULL



PostGIS 2.2.0dev 开发手册

207 / 686

**注意**

需要PostGIS安装时添加Proj的编译支持，使用函数PostGIS\_Full\_Version确认你是否安装了该库

in.

**注意**

可用版本: 1.2.2 –带有version参数的函数形式从1.3.2版本开始支持

**注意**

版本提升: 2.0.0 –添加namespace prefix（默认为kml）的支持，默认没有prefix

**注意**

版本变更: 2.0.0 –开始支持默认参数，以及命名参数

**注意**

函数ST\_AsKML 不支持没有SRID的几何对象

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

**样例**

SELECT ST\_AsKML(ST\_GeomFromText(’POLYGON((0 0,0 1,1 1,1 0,0 0))’,4326));

st\_askml

--------

<Polygon><outerBoundaryIs><LinearRing><coordinates>0,0 0,1 1,1 1,0 0,0</coordinates></ ←

LinearRing></outerBoundaryIs></Polygon>

--3d linestring

SELECT ST\_AsKML(’SRID=4326;LINESTRING(1 2 3, 4 5 6)’);

<LineString><coordinates>1,2,3 4,5,6</coordinates></LineString>

译者注：带有前缀的样例

**SELECT** ST\_AsKML(2,ST\_GeomFromText('POLYGON((0 0,0 1.123456789123456789,1 1,1 0,0 0))',4326),15,'AAAA');

<AAAA:Polygon>

<AAAA:outerBoundaryIs>

<AAAA:LinearRing>

<AAAA:coordinates>0,0 0,1.123456789123457 1,1 1,0 0,0</AAAA:coordinates>

</AAAA:LinearRing>

</AAAA:outerBoundaryIs>

</AAAA:Polygon>

**请参考**

ST\_AsSVG, ST\_AsGML

**8.7.8**

**ST\_AsSVG**

ST\_AsSVG — 根据给定geometry或geography对象，在一个SVG 数据中返回一个geometry对象。



PostGIS 2.2.0dev 开发手册

208 / 686

**用法**

text ST\_AsSVG(geometry geom, integer rel=0, integer maxdecimaldigits=15);

text ST\_AsSVG(geography geog, integer rel=0, integer maxdecimaldigits=15);

**描述**

在可伸缩矢量图形（简称SVG）中返回geometry对象。设置1时候表示使用相对移动的方式，默认即0是使用绝对移动的方式。第三个参数可以用来减少精度默认是小数点后面保留15位小数。当参数rel设置为0时候，Point几何对象会被拆分成cx/cy的格式;设置为1的时候，拆分成x/y的格式。多个点的几何对象需要用逗号“,”分隔， GeometryCollection 几何对象需要用分号分隔”;”

**注意**

可用版本: 1.2.2.

可用版本: 1.4.0 从1.4.0版本开始，在绝对路径中添加了L命令来保持和SVG规范一致，参考<http://www.w3.org/TR/SVG/paths.html#PathDataBNF>

版本变更: 2.0.0开始使用默认参数，并支持命名参数

**样例**

SELECT ST\_AsSVG(ST\_GeomFromText(’POLYGON((0 0,0 1,1 1,1 0,0 0))’,4326));

st\_assvg

--------

M 0 0 L 0 -1 1 -1 1 0 Z

**8.7.9**

**ST\_AsX3D**

ST\_AsX3D — 以X3D xml节点元素返回返回一个几何对象，格式遵守: ISO-IEC-19776-1.2-X3DEncodings-XML

**用法**

text ST\_AsX3D(geometry g1, integer maxdecimaldigits=15, integer options=0);

**描述**

以X3D xml节点元素返回返回一个几何对象，参考http://web3d.org/x3d/speciﬁcations/ISO-IEC-19776-1.2-X3DEncodings-

[XML/Part01/EncodingOfNodes.html.](http://web3d.org/x3d/specifications/ISO-IEC-19776-1.2-X3DEncodings-XML/Part01/EncodingOfNodes.html)如果没有指定精度，最多保留15位小数.

**注意**

上述函数有多个选项用来将PostGIS对象转换成X3D表示的方式。由于X3D几何类型表示和PostGIS几何类型没有直接对应，同时我们已经避免了一些更好地进行图像渲染的X3D类型，因为大部分渲染工具并不支持X3D的这些类型。这些类型关系映射是我们已经选定好的。如果你对我们怎样能让用户明白他们首选的映射问题有解决意见或者办法的话，我们会很乐意地放出一个 bug ticket（译者注：bug ticket就是bug任务单的意思，是bug管理工具）

下面是目前我们的PostGIS 2D/3D类型和X3D类型的对应列表



PostGIS 2.2.0dev 开发手册

209 / 686

PostGIS Type

2D X3D Type

3D X3D Type

**注意**

2D几何类型对象目前不支持，内环目前只是化成分离的polygon对象。我们正在对这些问题进行解决

在3D空间处理上面有很多进步，特别是HTML5与X3D的集成上 [X3D Integration with HTML5](http://www.web3d.org/x3d/wiki/index.php/X3D_and_HTML5#Goals:_X3D_and_HTML5)

有一个开源的X3D阅览器可以用来看渲染的几何对象，请查看. Free Wrl<http://freewrl.sourceforge.net/>

有Mac，Linux和Windows的二进制包。使用打包的FreeWRL\_Launcher 来查看几何对象

可用版本: 2.0.0: ISO-IEC-19776-1.2-X3DEncodings-XML

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

该函数支持 Polyhedral Surface类型几何对象.

该函数支持 Triangles 和 Triangulated Irregular Network Surfaces (TIN)类型.

**样例:创建一个完整功能的X3D文档-这会生成一个可以在FreeWrl 和 other X3D阅读器查看的立方体**

SELECT ’<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE X3D PUBLIC "ISO//Web3D//DTD X3D 3.0//EN" "http://www.web3d.org/specifications/x3d ←

-3.0.dtd">

<X3D>

<Scene>

<Transform>

<Shape>

<Appearance>

<Material emissiveColor=’’0 0 1’’/>

</Appearance> ’ ||

ST\_AsX3D( ST\_GeomFromEWKT(’POLYHEDRALSURFACE( ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)),

((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),

((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),

((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)) )’)) ||

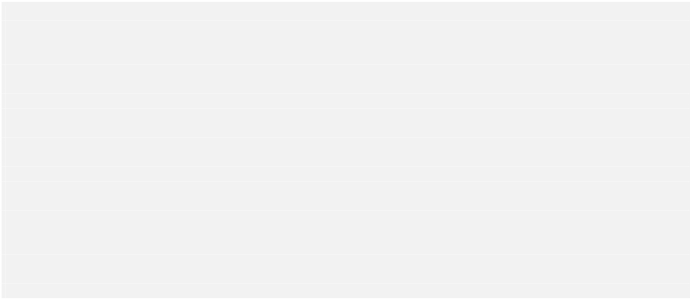
’</Shape>

</Transform>

</Scene>

</X3D>’ As x3ddoc;

x3ddoc



|  |  |  |
| --- | --- | --- |
| PostGIS Type | 2D X3D Type | 3D X3D Type |
| LINESTRING | 还没有实现，将会是  PolyLine2D类型 | LineSet |
| MULTILINESTRING | 还没有实现，将会是  PolyLine2D类型 | IndexedLineSet |
| MULTIPOINT | Polypoint2D | PointSet |
| POINT | 输出空格分隔的坐标 | 输出空格分隔的坐标 |
| (MULTI) POLYGON,  POLYHEDRALSURFACE | 无效的 X3D 标记 | IndexedFaceSet (内环目前输出的是其他面的集合) |
| TIN | TriangleSet2D (还没有实现) | IndexedTriangleSet |

PostGIS 2.2.0dev 开发手册

210 / 686

--------

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE X3D PUBLIC "ISO//Web3D//DTD X3D 3.0//EN" "http://www.web3d.org/specifications/x3d ←

-3.0.dtd">

<X3D>

<Scene>

<Transform>

<Shape>

<Appearance>

<Material emissiveColor=’0 0 1’/>

</Appearance>

<IndexedFaceSet

coordIndex=’0 1 2 3 -1 4 5 6 7 -1 8 9 10 11 -1 12 13 14 15 -1 16 17 ←

18 19 -1 20 21 22 23’>

<Coordinate point=’0 0 0 0 0 1 0 1 1 0 1 0 0 0 0 0 1 0 1 1 0 1 0 0 0 0 0 1 0 0

1 0 1 0 0 1 1 1 0 1 1 1 1 0 1 1 0 0 0 1 0 0 1 1 1 1 1 1 1 0 0 0 1 1 0 1 1 1

1 0 1 1’ />

</IndexedFaceSet>

</Shape>

</Transform>

</Scene>

</X3D>

**样例: An Octagon elevated 3 Units and decimal precision of 6**

SELECT ST\_AsX3D(

ST\_Translate(

ST\_Force\_3d(

ST\_Buffer(ST\_Point(10,10),5, ’quad\_segs=2’)), 0,0,

3)

,6) As x3dfrag;

x3dfrag

--------

<IndexedFaceSet coordIndex="0 1 2 3 4 5 6 7">

←

←

<Coordinate point="15 10 3 13.535534 6.464466 3 10 5 3 6.464466 6.464466 3 5 10 3

6.464466 13.535534 3 10 15 3 13.535534 13.535534 3 " />

</IndexedFaceSet>

**样例: TIN**

SELECT ST\_AsX3D(ST\_GeomFromEWKT(’TIN (((

0 0 0,

0 0 1,

0 1 0,

0 0 0

)), ((

0 0 0,

0 1 0,

1 1 0,

0 0 0

))

)’)) As x3dfrag;

x3dfrag

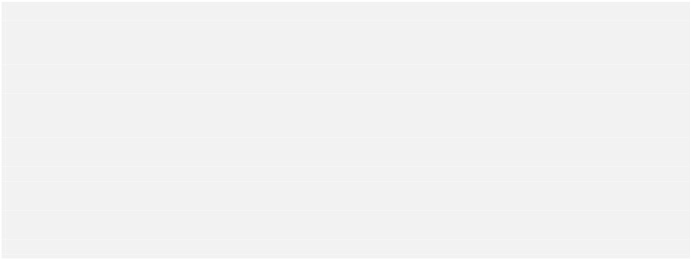
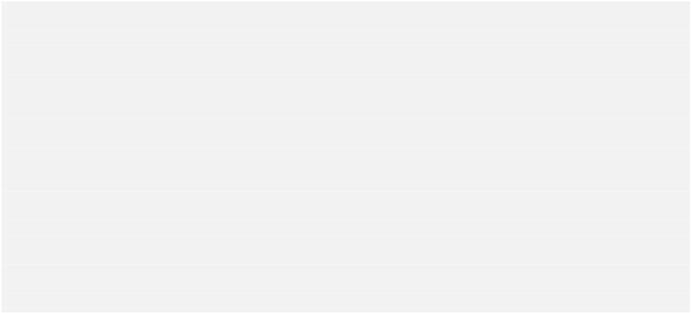
--------

←

<IndexedTriangleSet

index=’0 1 2 3 4 5’><Coordinate point=’0 0 0 0 0 1 0 1 0 0 0 0 0 1 0 1 ←

1 0’/></IndexedTriangleSet>



PostGIS 2.2.0dev 开发手册

211 / 686

**样例: Closed multilinestring (the boundary of a polygon with holes)**

SELECT ST\_AsX3D(

ST\_GeomFromEWKT(’MULTILINESTRING((20 0 10,16 -12 10,0 -16 10,-12 -12 10,-20 0

10,-12 16 10,0 24 10,16 16 10,20 0 10),

(12 0 10,8 8 10,0 12 10,-8 8 10,-8 0 10,-8 -4 10,0 -8 10,8 -4 10,12 0 10))’)

) As x3dfrag;

x3dfrag

--------

←

<IndexedLineSet

coordIndex=’0 1 2 3 4 5 6 7 0 -1 8 9 10 11 12 13 14 15 8’>

<Coordinate point=’20 0 10 16 -12 10 0 -16 10 -12 -12 10 -20 0 10 -12 16 10 0 24 10 16

16 10 12 0 10 8 8 10 0 12 10 -8 8 10 -8 0 10 -8 -4 10 0 -8 10 8 -4 10 ’ />

</IndexedLineSet>

←

**8.7.10**

**ST\_GeoHash**

ST\_GeoHash —返回一个几何对象的GeoHash表示

**用法**

text ST\_GeoHash(geometry geom, integer maxchars=full\_precision\_of\_point);

**描述**

返回一个几何对象的GeoHash表示(<http://en.wikipedia.org/wiki/Geohash>).一个GeoHash把一个point对象加密成一个基于前缀的可排序可收缩的形式。对一个点来说， GeoHash字符串越短，精度越低。GeoHash甚至可以被看做一个含有实际点的box对象。如果maxchars参数没有指定， ST\_GeoHash 函数返回一个输入对象全精度的GeoHash值。Point类型对象返回一个20个字符的GeoHash字符串（基本上能够保持输入对象的精度）。其他类型根据对象的大小返回一个可变精度的GeoHash字符串.更大的特征意味着更低的精度，更小的特征对象意味着精度更高（译者注：这里有点像分辨率，像点越大分辨率越低，像点越小分辨率越高）。这里的理念是用GeoHash表示的box总数包含输入的对象。

如果maxchars参数被指定， ST\_GeoHash函数会返回一个不超过指定maxchars值个数的字符来表示一个较低精度的几何对象。对于非point类型对象，计算的起始点是几何对象的bounding box的中心点

可用版本: 1.4.0

**注意**

函数ST\_GeoHash 不支持不在地理坐标系（坐标是经纬度格式）中的对象

该函数支持CircularString和Curve几何类型对象

**样例**

SELECT ST\_GeoHash(ST\_SetSRID(ST\_MakePoint(-126,48),4326));

st\_geohash

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

c0w3hf1s70w3hf1s70w3



PostGIS 2.2.0dev 开发手册

212 / 686

SELECT ST\_GeoHash(ST\_SetSRID(ST\_MakePoint(-126,48),4326),5);

st\_geohash

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

c0w3h

**请参考**

ST\_GeomFromGeoHash

**8.7.11**

**ST\_AsText**

ST\_AsText —返回不含有SRID信息的geometry或geography对象的WKT表示

**用法**

text ST\_AsText(geometry g1);

text ST\_AsText(geography g1);

**描述**

返回不含有SRID信息的geometry或geography对象的WKT表示

**注意**

WKT SPEC规范不包含SRID参数，使用非标准的（扩展的）函数ST\_AsEWKT返回带有SRID的几何对象

WKT 格式不维持对象的精度，因此为了阻止精度损失，使用函数ST\_AsBinary 或函数 ST\_AsEWKB

**注意**

对几何对象来说，函数ST\_AsText是函数ST\_GeomFromText逆功能函数，使用函数ST\_GeomFromText可以把一个ST\_AsText表述的几何对象转化成一个postgis的几何对象

可用版本: 1.5 -引入支持geography对象.

该函数方法实现了规范 OpenGIS Simple Features Implementation Speciﬁcation for SQL 1.1. s2.1.1.1

该函数方法实现了规范 SQL/MM speciﬁcation. SQL-MM 3: 5.1.25

该函数支持CircularString和Curve几何类型对象



PostGIS 2.2.0dev 开发手册

213 / 686

**样例**

SELECT ST\_AsText(’01030000000100000005000000000000000000

000000000000000000000000000000000000000000000000

F03F000000000000F03F000000000000F03F000000000000F03

F000000000000000000000000000000000000000000000000’);

st\_astext

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

POLYGON((0 0,0 1,1 1,1 0,0 0))

(1 row)

**请参考**

ST\_AsBinary, ST\_AsEWKB, ST\_AsEWKT, ST\_GeomFromText

**8.7.12**

**ST\_AsLatLonText**

ST\_AsLatLonText — 返回一个给定的度、分、秒表示

**用法**

text ST\_AsLatLonText(geometry pt);

text ST\_AsLatLonText(geometry pt, text format);

**描述**

返回一个给定的度、分、秒表示

**注意**

该函数假定点是处于lat/lon投影系中。X（表示经度）坐标和Y（表示维度）坐标会在输出进行正则化到正常范围内（经度范围是：-180到180，维度范围是-90到90）

该函数的第二种形式的第二个参数format参数，是一个返回文本格式的字符串，与时间格式字符串很类似。有效的标记意义如下：D表示degree，即度；M表示minute，分；C表示主方向（NSEW）。DMS标志可以重复来表示所需的宽度和精度例如（SSS.SSSS表示1.0023）。

标志格式的参数"M", "S", 和 "C" 都是可选的. 如果参数C被省略了，度数用符合“-”表示。如果参数S被省略了，minute值会被显示成你定义的任意精度的小数。如果参数M被省略了，度数被显示成你定义的任意精度小数。如果输出格式的字符串被省略了（或零长度），那么将使用默认的输出格式。

可用版本: 2.0

**样例**

Default format.

SELECT (ST\_AsLatLonText(’POINT (-3.2342342 -2.32498)’));

st\_aslatlontext

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

2\textdegree{}19’29.928"S 3\textdegree{}14’3.243"W



PostGIS 2.2.0dev 开发手册

214 / 686

Providing a format (same as the default).

SELECT (ST\_AsLatLonText(’POINT (-3.2342342 -2.32498)’, ’D\textdegree{}M’’S.SSS"C’));

st\_aslatlontext

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

2\textdegree{}19’29.928"S 3\textdegree{}14’3.243"W

Characters other than D, M, S, C and . are just passed through.

SELECT (ST\_AsLatLonText(’POINT (-3.2342342 -2.32498)’, ’D degrees, M minutes, S seconds to

the C’));

st\_aslatlontext

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

2 degrees, 19 minutes, 30 seconds to the S 3 degrees, 14 minutes, 3 seconds to the W

Signed degrees instead of cardinal directions.

SELECT (ST\_AsLatLonText(’POINT (-3.2342342 -2.32498)’, ’D\textdegree{}M’’S.SSS"’));

st\_aslatlontext

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

-2\textdegree{}19’29.928" -3\textdegree{}14’3.243"

Decimal degrees.

SELECT (ST\_AsLatLonText(’POINT (-3.2342342 -2.32498)’, ’D.DDDD degrees C’));

st\_aslatlontext

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

2.3250 degrees S 3.2342 degrees W

Excessively large values are normalized.

SELECT (ST\_AsLatLonText(’POINT (-302.2342342 -792.32498)’));

st\_aslatlontext

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

72\textdegree{}19’29.928"S 57\textdegree{}45’56.757"E

←

**8.7.13**

**ST\_AsTWKB**

ST\_AsTWKB —输出TWKB（即Tiny WKB）描述的几何对象

**用法**

bytea ST\_AsTWKB(geometry g1, integer decimaldigits);

bytea ST\_AsTWKB(geometry g1, integer decimaldigits, int8 geometryID);

**描述**

返回TWKB描述的几何对象。TWKB是一种新型的压缩二进制格式。第二个整型参数用来定义输出坐标的小数位数。第三个参数是可选的，用于给每个TWKB几何对象设定一个唯一的ID

**注意**

TWKB 依然是一个正在发展的规范。格式描述如下here（https://github.com/nicklasaven/TWKB） ,生成客户端代码可以在这里找到[here](https://github.com/nicklasaven/twkb_web)

可用版本: 2.2.0



PostGIS 2.2.0dev 开发手册

215 / 686

**样例**

SELECT ST\_AsTWKB(’LINESTRING(1 1,5 5)’::geometry,0,1);

st\_astwkb

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

\x0342010202020808

**请参考**

ST\_AsTWKBAgg, ST\_AsBinary, ST\_AsEWKB, ST\_AsEWKT, ST\_GeomFromText

**8.7.14**

**ST\_AsTWKBAgg**

ST\_AsTWKBAgg — 聚合一些几何对象，并返回TWKB格式的描述对象

**用法**

bytea ST\_AsTWKBAgg(geometry set g1, integer decimaldigits);

bytea ST\_AsTWKBAgg(geometry set g1, integer decimaldigits, int8 geometryID);

**描述**

返回TWKB描述的几何对象。TWKB是一种新型的压缩二进制格式。

这是ST\_AsTWKB函数的聚合函数版本。它聚集一些几何对象，并返回TWKB描述的几何对象。返回结果中，每个输入的几何对象都存储一个唯一的ID。第二个参数定义输出时候坐标的小数点位数。第三个参数是可选的，用来给每个TWKB几何对象一个唯一的ID

**注意**

TWKB 依然是一个正在发展的规范。格式描述如下here（https://github.com/nicklasaven/TWKB） ,生成客户端代码可以在这里找到[here](https://github.com/nicklasaven/twkb_web)

可用版本: 2.2.0

**样例**

SELECT ST\_AsTWKBAgg(geom,0,id) FROM

(SELECT ’LINESTRING(1 1,5 5)’::geometry geom, 1 AS id

UNION ALL

SELECT ’LINESTRING(6 5, 1 7)’::geometry AS geom, 2 AS id) foo;

st\_astwkbagg

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

\x035602010202020808020202000904

**请参考**

ST\_AsTWKB, ST\_AsBinary, ST\_AsEWKB, ST\_AsEWKT, ST\_GeomFromText



PostGIS 2.2.0dev 开发手册

216 / 686

**8.8**

**8.8.1**

**几何对象操作运算符**

**&&**

&& — 如果输入对象A的2D bounding box与输入对象B的2D bounding box有交集，则返回TRUE

**用法**

boolean &&( geometry A , geometry B );

boolean &&( geography A , geography B );

**描述**

如果输入对象A的2D bounding box与输入对象B的2D bounding box有交集，则返回TRUE

**注意**

该操作符会利用几何对象上任何可用的索引

版本提升: 2.0.0 版本开始支持Polyhedral Surface几何对象.

可用版本: 1.5.0引入支持geography对象.

该函数支持CircularString和Curve几何类型对象

该函数支持 Polyhedral Surface类型几何对象.

**样例**

SELECT tbl1.column1, tbl2.column1, tbl1.column2 && tbl2.column2 AS overlaps

FROM ( VALUES

(1, ’LINESTRING(0 0, 3 3)’::geometry),

(2, ’LINESTRING(0 1, 0 5)’::geometry)) AS tbl1,

( VALUES

(3, ’LINESTRING(1 2, 4 6)’::geometry)) AS tbl2;

column1 | column1 | overlaps

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

1 | 3 | t

2 | 3 | f

(2 rows)

**请参考**

|&>,&>,&<|,&<, ~, @

**8.8.2**

**&&&**

&&& —如果输入对象A的n-D bounding box与输入对象B的n-D bounding box有交集，则返回TRUE



PostGIS 2.2.0dev 开发手册

217 / 686

**用法**

boolean &&&( geometry A , geometry B );

**描述**

如果输入对象A的n-D bounding box与输入对象B的n-D bounding box有交集，则返回TRUE.

**注意**

该运算符可以利用几何对象上任意索引

可用版本: 2.0.0

该函数支持CircularString和Curve几何类型对象

该函数支持 Polyhedral Surface类型几何对象.

该函数支持 Triangles 和 Triangulated Irregular Network Surfaces (TIN)类型.

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

**样例: 3D LineStrings**

SELECT tbl1.column1, tbl2.column1, tbl1.column2 &&& tbl2.column2 AS overlaps\_3d,

tbl1.column2 && tbl2.column2 AS overlaps\_2d

FROM ( VALUES

(1, ’LINESTRING Z(0 0 1, 3 3 2)’::geometry),

(2, ’LINESTRING Z(1 2 0, 0 5 -1)’::geometry)) AS tbl1,

( VALUES

(3, ’LINESTRING Z(1 2 1, 4 6 1)’::geometry)) AS tbl2;

column1 | column1 | overlaps\_3d | overlaps\_2d

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

1 | 3 | t

2 | 3 | f

| t

| t

**样例: 3M LineStrings**

SELECT tbl1.column1, tbl2.column1, tbl1.column2 &&& tbl2.column2 AS overlaps\_3zm,

tbl1.column2 && tbl2.column2 AS overlaps\_2d

FROM ( VALUES

(1, ’LINESTRING M(0 0 1, 3 3 2)’::geometry),

(2, ’LINESTRING M(1 2 0, 0 5 -1)’::geometry)) AS tbl1,

( VALUES

(3, ’LINESTRING M(1 2 1, 4 6 1)’::geometry)) AS tbl2;

column1 | column1 | overlaps\_3zm | overlaps\_2d

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

1 | 3 | t

2 | 3 | f

| t

| t



PostGIS 2.2.0dev 开发手册

218 / 686

**请参考**

&&

**8.8.3**

**&<**

&< — 如果输入几何对象A的bounding box与输入几何对象B的bounding box重合或者在后者的右边，则返回TRUE

**用法**

boolean &<( geometry A , geometry B );

**描述**

如果输入几何对象A的bounding box与输入几何对象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，覆盖或者不在输入几何对象B 的bounding box的右侧。

**注意**

该运算符可以利用几何对象上任意索引

**样例**

SELECT tbl1.column1, tbl2.column1, tbl1.column2 &< tbl2.column2 AS overleft

FROM

( VALUES

(1, ’LINESTRING(1 2, 4 6)’::geometry)) AS tbl1,

( VALUES

(2, ’LINESTRING(0 0, 3 3)’::geometry),

(3, ’LINESTRING(0 1, 0 5)’::geometry),

(4, ’LINESTRING(6 0, 6 1)’::geometry)) AS tbl2;

column1 | column1 | overleft

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

1 | 2 | f

1 | 3 | f

1 | 4 | t

(3 rows)

**请参考**

&&, |&>,&>,&<|

**8.8.4**

**&<|**

&<| — 如果A的bounding box覆盖或在B的bounding box的下侧，则返回TRUE

**用法**

boolean &<|( geometry A , geometry B );



PostGIS 2.2.0dev 开发手册

219 / 686

**描述**

如果A的bounding box覆盖或在B的bounding box的下侧，则返回TRUE，或者更准确地说覆盖或者不在输入几何对象B的bounding box的上侧

该函数支持CircularString和Curve几何类型对象

该函数支持 Polyhedral Surface类型几何对象.

**注意**

该运算符可以利用几何对象上任意索引

**样例**

SELECT tbl1.column1, tbl2.column1, tbl1.column2 &<| tbl2.column2 AS overbelow

FROM

( VALUES

(1, ’LINESTRING(6 0, 6 4)’::geometry)) AS tbl1,

( VALUES

(2, ’LINESTRING(0 0, 3 3)’::geometry),

(3, ’LINESTRING(0 1, 0 5)’::geometry),

(4, ’LINESTRING(1 2, 4 6)’::geometry)) AS tbl2;

column1 | column1 | overbelow

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

1 | 2 | f

1 | 3 | t

1 | 4 | t

(3 rows)

**请参考**

&&, |&>,&>,&<

**8.8.5**

**&>**

&> —如果A的bounding box覆盖或在B的bounding box的右侧，则返回TRUE

**用法**

boolean &>( geometry A , geometry 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输入几何对象B的bounding box的左侧

**注意**

该运算符可以利用几何对象上任意索引



PostGIS 2.2.0dev 开发手册

220 / 686

**样例**

SELECT tbl1.column1, tbl2.column1, tbl1.column2 &> tbl2.column2 AS overright

FROM

( VALUES

(1, ’LINESTRING(1 2, 4 6)’::geometry)) AS tbl1,

( VALUES

(2, ’LINESTRING(0 0, 3 3)’::geometry),

(3, ’LINESTRING(0 1, 0 5)’::geometry),

(4, ’LINESTRING(6 0, 6 1)’::geometry)) AS tbl2;

column1 | column1 | overright

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

1 | 2 | t

1 | 3 | t

1 | 4 | f

(3 rows)

**请参考**

&&, |&>,&<|,&<

**8.8.6**

**<<**

<< — 如果A的bounding box严格地在B的bounding box左侧，则返回TRUE

**用法**

boolean <<( geometry A , geometry B );

**描述**

如果A的bounding box严格地在B的bounding box左侧，则返回TRUE

**注意**

该运算符可以利用几何对象上任意索引

**样例**

SELECT tbl1.column1,

FROM

( VALUES

(1, ’LINESTRING (1

( VALUES

(2, ’LINESTRING (0

(3, ’LINESTRING (6

(4, ’LINESTRING (2

tbl2.column1, tbl1.column2 << tbl2.column2 AS left

2, 1 5)’::geometry)) AS tbl1,

0, 4 3)’::geometry),

0, 6 5)’::geometry),

2, 5 6)’::geometry)) AS tbl2;

column1 | column1 | left

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

1 | 2 | f

1 | 3 | t

1 | 4 | t

(3 rows)



PostGIS 2.2.0dev 开发手册

221 / 686

**请参考**

>>, |>>,<<|

**8.8.7**

**<<|**

<<| —如果A的bounding box严格地在B的bounding box下侧，则返回TRUE

**用法**

boolean <<|( geometry A , geometry B );

**描述**

如果A的bounding box严格地在B的bounding box下侧，则返回TRUE

**注意**

该运算符可以利用几何对象上任意索引

**样例**

SELECT tbl1.column1,

FROM

( VALUES

(1, ’LINESTRING (0

( VALUES

tbl2.column1, tbl1.column2 <<| tbl2.column2 AS below

0, 4 3)’::geometry)) AS tbl1,

(2,

LINESTRING (1

4, 1 7)’::geometry),

(3, ’LINESTRING (6

(4, ’LINESTRING (2

1, 6 5)’::geometry),

3, 5 6)’::geometry)) AS tbl2;

column1 | column1 | below

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

1 | 2 | t

1 | 3 | f

1 | 4 | f

(3 rows)

**请参考**

<<,>>, |>>

**8.8.8**

**=**

= —如果A的bounding box与B的bounding box相同，则返回TRUE。注意：输入的A和B几何对象的bounding box都是双精度

**用法**

boolean =( geometry A , geometry B );

boolean =( geography A , geography B );



PostGIS 2.2.0dev 开发手册

222 / 686

**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使用针对几何对象定义的运算符=，<和>来进行内部排序和几何对象的比较（例如在GROUP BY或ORDER BY这样的SQL子句中）

**警告**

该函数引起一些混淆。当你进行比较geometry对象A=geometry对象B适合，该等式会返回TRUE，即便这两个几何对象是明显不同的，但却有相同的bounding box。使用函数ST\_OrderingEqual或函数ST\_Equal来检验者两个几何对象是否是真的相同

**注意**

该运算符不会利用几何对象上任意索引

该函数支持CircularString和Curve几何类型对象

该函数支持 Polyhedral Surface类型几何对象.

版本变更: 2.0.0 , 从该版本开始bounding box改变为使用双精度，而不再使用float4单精度的。这方面的副作用是，在以前版本中特定的点可能会有稍微不同。在之前的版本函数可能返回true，而在2.0+版本中将返回false。因为它们的float 4精度的bounding box是相同的，但float8精度下，bounding box将会不同

**样例**

SELECT ’LINESTRING(0 0, 0 1, 1 0)’::geometry = ’LINESTRING(1 1, 0 0)’::geometry;

?column?

----------

t

(1 row)

SELECT ST\_AsText(column1)

FROM ( VALUES

(’LINESTRING(0 0, 1 1)’::geometry),

(’LINESTRING(1 1, 0 0)’::geometry)) AS foo;

st\_astext

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

LINESTRING(0 0,1 1)

LINESTRING(1 1,0 0)

(2 rows)

-- 注意: the GROUP BY uses the "=" to compare for geometry equivalency.

SELECT ST\_AsText(column1)

FROM ( VALUES

(’LINESTRING(0 0, 1 1)’::geometry),

(’LINESTRING(1 1, 0 0)’::geometry)) AS foo

GROUP BY column1;

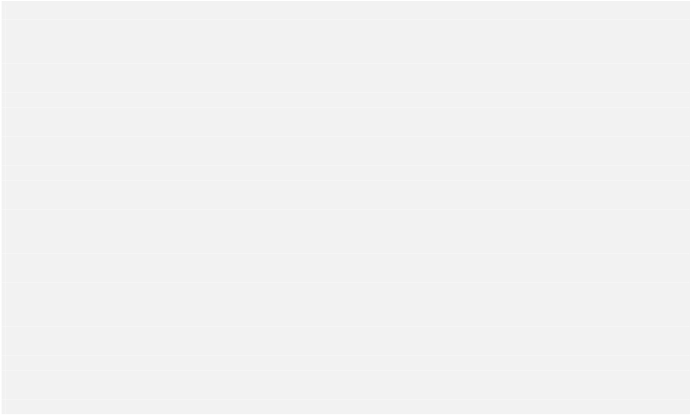
st\_astext

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

LINESTRING(0 0,1 1)

(1 row)

-- In versions prior to 2.0, this used to return true --



PostGIS 2.2.0dev 开发手册

223 / 686

SELECT ST\_GeomFromText(’POINT(1707296.37 4820536.77)’) =

ST\_GeomFromText(’POINT(1707296.27 4820536.87)’) As pt\_intersect;

--pt\_intersect --

f

**请参考**

ST\_Equals, ST\_OrderingEquals

**8.8.9**

**>>**

>> — 如果几何对象A的bounding box严格在几何对象B的右边，则返回true

**用法**

boolean >>( geometry A , geometry B );

**描述**

如果几何对象A的bounding box严格在几何对象B的右边，则返回true

**注意**

该运算符可以利用几何对象上任意索引

**样例**

SELECT tbl1.column1,

FROM

( VALUES

(1, ’LINESTRING (2

( VALUES

(2, ’LINESTRING (1

(3, ’LINESTRING (6

(4, ’LINESTRING (0

tbl2.column1, tbl1.column2 >> tbl2.column2 AS right

3, 5 6)’::geometry)) AS tbl1,

4, 1 7)’::geometry),

1, 6 5)’::geometry),

0, 4 3)’::geometry)) AS tbl2;

column1 | column1 | right

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

1 | 2 | t

1 | 3 | f

1 | 4 | f

(3 rows)

**请参考**

<<, |>>,<<|

**8.8.10**

**@**

@ —如果几何对象A的bounding box被几何对象B的bounding box严格包含，则返回true



PostGIS 2.2.0dev 开发手册

224 / 686

**用法**

boolean @( geometry A , geometry B );

**描述**

如果几何对象A的bounding box被几何对象B的bounding box严格包含，则返回true

**注意**

该运算符可以利用几何对象上任意索引

**样例**

SELECT tbl1.column1,

FROM

( VALUES

(1, ’LINESTRING (1

( VALUES

(2, ’LINESTRING (0

(3, ’LINESTRING (2

(4, ’LINESTRING (1

tbl2.column1, tbl1.column2 @ tbl2.column2 AS contained

1, 3 3)’::geometry)) AS tbl1,

0, 4 4)’::geometry),

2, 4 4)’::geometry),

1, 3 3)’::geometry)) AS tbl2;

column1 | column1 | contained

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

1 | 2 | t

1 | 3 | f

1 | 4 | t

(3 rows)

**请参考**

~,&&

**8.8.11**

**|&>**

|&> —如果几何对象A的bounding box覆盖B的bounding box或在B的bounding box上面，则返回true

**用法**

boolean |&>( geometry A , geometry B );

**描述**

如果几何对象A的bounding box覆盖B的bounding box或在B的bounding box上面，则返回true，或者更精确的说覆盖B的bounding box或者不在B的bounding box下面。

**注意**

该运算符可以利用几何对象上任意索引



PostGIS 2.2.0dev 开发手册

225 / 686

**样例**

SELECT tbl1.column1, tbl2.column1, tbl1.column2 |&> tbl2.column2 AS overabove

FROM

( VALUES

(1, ’LINESTRING(6 0, 6 4)’::geometry)) AS tbl1,

( VALUES

(2, ’LINESTRING(0 0, 3 3)’::geometry),

(3, ’LINESTRING(0 1, 0 5)’::geometry),

(4, ’LINESTRING(1 2, 4 6)’::geometry)) AS tbl2;

column1 | column1 | overabove

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

1 | 2 | t

1 | 3 | f

1 | 4 | f

(3 rows)

**请参考**

&&,&>,&<|,&<

**8.8.12**

**|>>**

|>> — 如果A的bounding box严格地在B的bounding box上面，则返回TRUE

**用法**

boolean |>>( geometry A , geometry B );

**描述**

如果A的bounding box严格地在B的bounding box上面，则返回TRUE

**注意**

该运算符可以利用几何对象上任意索引

**样例**

SELECT tbl1.column1,

FROM

( VALUES

(1, ’LINESTRING (1

( VALUES

(2, ’LINESTRING (0

(3, ’LINESTRING (6

(4, ’LINESTRING (2

tbl2.column1, tbl1.column2 |>> tbl2.column2 AS above

4, 1 7)’::geometry)) AS tbl1,

0, 4 2)’::geometry),

1, 6 5)’::geometry),

3, 5 6)’::geometry)) AS tbl2;

column1 | column1 | above

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

1 | 2 | t

1 | 3 | f

1 | 4 | f

(3 rows)



PostGIS 2.2.0dev 开发手册

226 / 686

**请参考**

<<,>>,<<|

**8.8.13**

**~**

~ —如果A的bounding box包含B的bounding box，则返回TRUE

**用法**

boolean ~( geometry A , geometry B );

**描述**

如果A的bounding box包含B的bounding box，则返回TRUE

**注意**

该运算符可以利用几何对象上任意索引

**样例**

SELECT tbl1.column1,

FROM

( VALUES

(1, ’LINESTRING (0

( VALUES

(2, ’LINESTRING (0

(3, ’LINESTRING (1

(4, ’LINESTRING (0

tbl2.column1, tbl1.column2 ~ tbl2.column2 AS contains

0, 3 3)’::geometry)) AS tbl1,

0, 4 4)’::geometry),

1, 2 2)’::geometry),

0, 3 3)’::geometry)) AS tbl2;

column1 | column1 | contains

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

1 | 2 | f

1 | 3 | t

1 | 4 | t

(3 rows)

**请参考**

@,&&

**8.8.14**

**~=**

~= —如果A的bounding box和B的bounding box相同，则返回TRUE

**用法**

boolean ~=( geometry A , geometry B );



PostGIS 2.2.0dev 开发手册

227 / 686

**描述**

如果A的bounding box和B的bounding box相同，则返回TRUE

**注意**

该运算符可以利用几何对象上任意索引

可用版本: 1.5.0 修改了方式

该函数支持 Polyhedral Surface类型几何对象.

**警告**

在PostGIS 1.5版本中，该运算符已经改变了实现（运算结果会和之前不同）,原本用于检测几何对象是否相等，现在只检测该几何对象的bounding box是否相同。复杂一点说，还依赖于你是否对你的数据库进行了硬升级或者软升级。你可以运行如下样例查询来确

认你的数据库是哪一种方式的实现。使用函数ST\_OrderingEquals或 ST\_Equals来验证真正的相等，而使用=来检查bounding box

是否相等。运算符=通常是更全的选择

**样例**

select ’LINESTRING(0 0, 1 1)’::geometry ~= ’LINESTRING(0 1, 1 0)’::geometry as equality;

equality

|

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

t

|

The above can be used to test if you have the new or old behavior of ~= operator.

**请参考**

ST\_Equals, ST\_OrderingEquals, =

**8.8.15**

**<->**

<-> —返回两个点的距离. 对于point或point类型检查，该运算符会使用浮点精度（相对于基本的几点对象所使用的double 双精度）。对于其他几何类型，返回的是bounding box的中心点之间的双精度输出距离。在距离排序和使用KNN 功能进行近邻限制输出对象个数时候很有用。（译者注：KNN是机器学习算法之一，被称作近邻算法，输出时候通常会限制输出多少个与输入点类似的对象，可以参考相关书籍）

**用法**

double precision <->( geometry A , geometry B );

**描述**

返回两个点的距离. 对于point或point类型检查，该运算符会使用浮点精度（相对于基本的几点对象所使用的double 双精度）。对于其他几何类型，返回的是bounding box的中心点之间的双精度输出距离。在距离排序和使用KNN 功能进行近邻限制输出对象个数时候很有用。（译者注：KNN是机器学习算法之一，被称作近邻算法，输出时候通常会限制输出多少个与输入点类似的对象，可以参考相关书籍）



PostGIS 2.2.0dev 开发手册

228 / 686

**注意**

该运算符可以利用几何对象上任意索引。该运算符和其他运算符不同的是，空间索引使用只能用于ORDER BY 子句的场景

**注意**

如果输入参数的其中一个几何对象是一个常量（不在子查询中），例如’SRID=3005;POINT(1011102450541)’::geometry则使用索引，如果运算符左边或右边均为普通的geometry，那么不会使用索引

参考资料，获取实用样例 OpenGeo workshop: Nearest-Neighbour Searching.

可用版本: 2.0.0 只支持PostgreSQL 9.1+

**样例**

SELECT ST\_Distance(geom, ’SRID=3005;POINT(1011102 450541)’::geometry) as d,edabbr, vaabbr

FROM va2005

ORDER BY d limit 10;

d

| edabbr | vaabbr

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

0 | ALQ

5541.57712511724 | ALQ

5579.67450712005 | ALQ

6083.4207708641 | ALQ

7691.2205404848 | ALQ

7900.75451037313 | ALQ

8694.20710669982 | ALQ

9564.24289057111 | ALQ

12089.665931705 | ALQ

18472.5531479404 | ALQ

| 128

| 129A

| 001

| 131

| 003

| 122

| 129B

| 130

| 127

| 002

(10 rows)

Then the KNN raw answer:

SELECT st\_distance(geom, ’SRID=3005;POINT(1011102 450541)’::geometry) as d,edabbr, vaabbr

FROM va2005

ORDER BY geom <-> ’SRID=3005;POINT(1011102 450541)’::geometry limit 10;

d

| edabbr | vaabbr

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

0 | ALQ

5579.67450712005 | ALQ

5541.57712511724 | ALQ

8694.20710669982 | ALQ

9564.24289057111 | ALQ

6083.4207708641 | ALQ

12089.665931705 | ALQ

24795.264503022 | ALQ

24587.6584922302 | ALQ

26764.2555463114 | ALQ

| 128

| 001

| 129A

| 129B

| 130

| 131

| 127

| 124

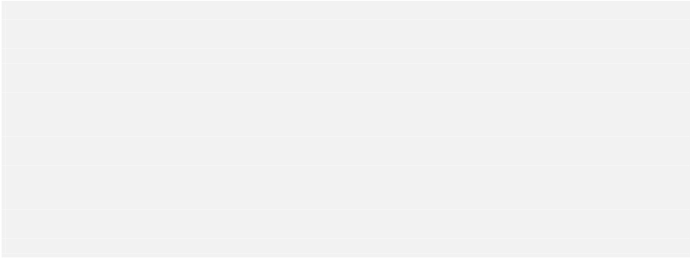
| 123

| 125

(10 rows)

Notice the misordering in the actual distances and the different entries that actually show up in the top 10.

Finally the hybrid:



PostGIS 2.2.0dev 开发手册

229 / 686

WITH index\_query AS (

SELECT ST\_Distance(geom, ’SRID=3005;POINT(1011102 450541)’::geometry) as d,edabbr, vaabbr

FROM va2005

ORDER BY geom <-> ’SRID=3005;POINT(1011102 450541)’::geometry LIMIT 100)

SELECT \*

FROM index\_query

ORDER BY d limit 10;

d

| edabbr | vaabbr

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

0 | ALQ

5541.57712511724 | ALQ

5579.67450712005 | ALQ

6083.4207708641 | ALQ

7691.2205404848 | ALQ

7900.75451037313 | ALQ

8694.20710669982 | ALQ

9564.24289057111 | ALQ

12089.665931705 | ALQ

18472.5531479404 | ALQ

(10 rows)

**请参考**

ST\_DWithin, ST\_Distance,<#>

| 128

| 129A

| 001

| 131

| 003

| 122

| 129B

| 130

| 127

| 002

**8.8.16**

**<#>**

<#> —返回两个几何对象的bounding box之间的距离。该运算符会使用浮点精度（相对于基本的几点对象所使用的double 双精度）。对于其他几何类型，返回的是bounding box的中心点之间的双精度输出距离。在距离排序和使用KNN 功能进行近邻限制输出对象个数时候很有用。（译者注：KNN是机器学习算法之一，被称作近邻算法，输出时候通常会限制输出多少个与输入点类似的对象，可以参考相关书籍

**用法**

double precision <#>( geometry A , geometry B );

**描述**

<#>运算符计算两个单精度的bounding box之间的距离，如果有空间索引，则使用空间索引。对于使用近邻计算后距离排序很有用

**注意**

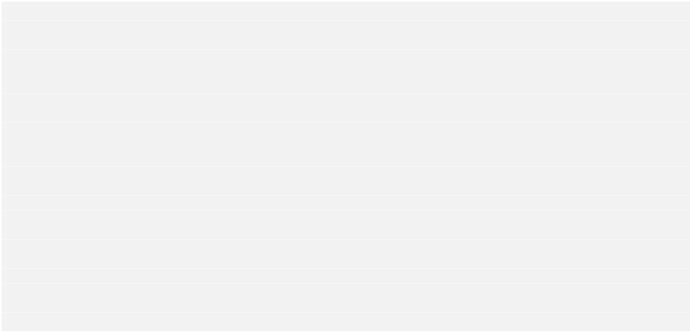
该运算符可以利用几何对象上任意索引。该运算符和其他运算符不同的是，空间索引使用只能用于ORDER BY 子句的场景

**注意**

该运算符只有在几何对象之一是常量的时候，例如ORDER BY (ST\_GeomFromText(’POINT(1 2)’) <#> geom)会使用索引，如果是

g1.geom <#>g2.geom则不使用索引

可用版本: 2.0.0 只支持PostgreSQL 9.1+版本的数据库



PostGIS 2.2.0dev 开发手册

230 / 686

**样例**

SELECT \*

FROM (

SELECT b.tlid, b.mtfcc,

b.geom <#> ST\_GeomFromText(’LINESTRING(746149 2948672,745954 2948576,

745787 2948499,745740 2948468,745712 2948438,

745690 2948384,745677 2948319)’,2249) As b\_dist,

ST\_Distance(b.geom, ST\_GeomFromText(’LINESTRING(746149 2948672,745954 2948576,

745787 2948499,745740 2948468,745712 2948438,

745690 2948384,745677 2948319)’,2249)) As act\_dist

FROM bos\_roads As b

ORDER BY b\_dist, b.tlid

LIMIT 100) As foo

ORDER BY act\_dist, tlid LIMIT 10;

tlid

| mtfcc |

b\_dist

|

act\_dist

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

85732027 | S1400 | 0 | 0

85732029 | S1400 | 0 | 0

85732031 | S1400 | 0 | 0

85734335 | S1400 | 0 | 0

85736037 | S1400 | 0 | 0

624683742 | S1400 | 0 | 128.528874268666

85719343 | S1400 | 260.839270432962 | 260.839270432962

85741826 | S1400 | 164.759294123275 | 260.839270432962

85732032 | S1400 | 277.75 | 311.830282365264

85735592 | S1400 | 222.25 | 311.830282365264

(10 rows)

**请参考**

ST\_DWithin, ST\_Distance,<->

**8.9**

**8.9.1**

**空间关系和测量函数**

**ST\_3DClosestPoint**

ST\_3DClosestPoint —返回3维内离g2几何对象最近的g1上的点。这个点也是g1和g2之间最短的线的第一个点

**用法**

geometry ST\_3DClosestPoint(geometry g1, geometry g2);

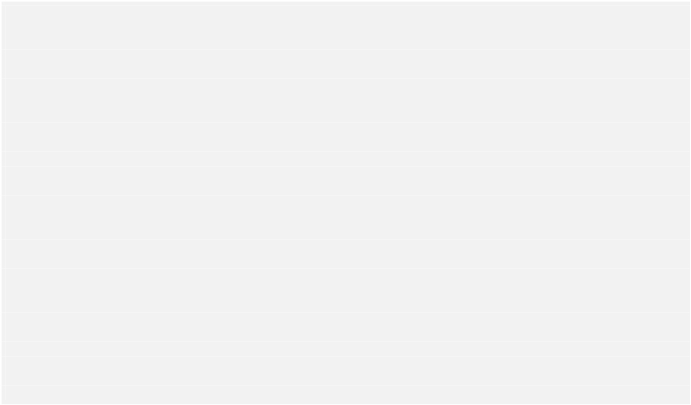
**描述**

返回3维内离g2几何对象最近的g1上的点。这个点也是g1和g2之间最短的线的第一个点。最短的3D线段的长度就是这两个几何对象之间的距离。

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

该函数支持 Polyhedral Surface类型几何对象.

可用版本: 2.0.0



PostGIS 2.2.0dev 开发手册

231 / 686

**样例**

linestring and point -- both 3d and 2d closest point

SELECT ST\_AsEWKT(ST\_3DClosestPoint(line,pt)) AS cp3d\_line\_pt,

ST\_AsEWKT(ST\_ClosestPoint(line,pt)) As cp2d\_line\_pt

FROM (SELECT ’POINT(100 100 30)’::geometry As pt,

’LINESTRING (20 80 20, 98 190 1, 110 180 3, 50 75 1000)’:: ←

geometry As line

) As foo;

cp3d\_line\_pt

cp2d\_line\_pt

|

←

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

POINT(54.6993798867619 128.935022917228 11.5475869506606) | POINT(73.0769230769231

115.384615384615)

linestring and multipoint -- both 3d and 2d closest point

SELECT ST\_AsEWKT(ST\_3DClosestPoint(line,pt)) AS cp3d\_line\_pt,

ST\_AsEWKT(ST\_ClosestPoint(line,pt)) As cp2d\_line\_pt

FROM (SELECT ’MULTIPOINT(100 100 30, 50 74 1000)’::geometry As pt,

’LINESTRING (20 80 20, 98 190 1, 110 180 3, 50 75 900)’:: ←

geometry As line

) As foo;

←

cp3d\_line\_pt

| cp2d\_line\_pt

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

POINT(54.6993798867619 128.935022917228 11.5475869506606) | POINT(50 75)

Multilinestring and polygon both 3d and 2d closest point

SELECT ST\_AsEWKT(ST\_3DClosestPoint(poly, mline)) As cp3d,

ST\_AsEWKT(ST\_ClosestPoint(poly, mline)) As cp2d

FROM (SELECT

ST\_GeomFromEWKT(’POLYGON((175 150 5, 20 40 5, 35 45 5, 50 60 5,

←

100 100 5, 175 150 5))’) As poly,

ST\_GeomFromEWKT(’MULTILINESTRING((175 155 2, 20 40 20, 50 60 -2, 125

100 1, 175 155 1),

(1 10 2, 5 20 1))’) As mline ) As foo;

←

cp3d

|

cp2d

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

POINT(39.993580415989 54.1889925532825 5) | POINT(20 40)

**请参考**

ST\_AsEWKT, ST\_ClosestPoint, ST\_3DDistance, ST\_3DShortestLine

**8.9.2**

**ST\_3DDistance**

ST\_3DDistance —对于3维的geometry类型对象，返回两个几何对象的最短笛卡尔距离(基于SRS表)，单位以投影系单位为准。

PostGIS 2.2.0dev 开发手册

232 / 686

**用法**

ﬂoat ST\_3DDistance(geometry g1, geometry g2);

**描述**

对于3维的geometry类型对象，返回两个几何对象的最短笛卡尔距离(基于SRS表)，单位以投影系单位为准。

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

该函数支持 Polyhedral Surface类型几何对象.

该函数方法实现了规范 SQL/MM speciﬁcation. SQL-MM ?

该方法由SFCGAL库提供后台支持.

可用版本: 2.0.0

**样例**

-- Geometry example - units in meters (SRID: 2163 US National Atlas Equal area) (3D point

←

and line compared 2D point and line)

-- 注意: currently no vertical datum support so Z is not transformed and assumed to be same ←

units as final.

SELECT ST\_3DDistance(

ST\_Transform(ST\_GeomFromEWKT(’SRID=4326;POINT(-72.1235 42.3521 4)’),2163),

ST\_Transform(ST\_GeomFromEWKT(’SRID=4326;LINESTRING(-72.1260 42.45 15, -72.123 42.1546 ←

20)’),2163)

) As dist\_3d,

ST\_Distance(

ST\_Transform(ST\_GeomFromText(’POINT(-72.1235 42.3521)’,4326),2163),

ST\_Transform(ST\_GeomFromText(’LINESTRING(-72.1260 42.45, -72.123 42.1546)’, 4326) ←

,2163)

) As dist\_2d;

dist\_3d

|

dist\_2d

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

127.295059324629 | 126.66425605671

-- Multilinestring and polygon both 3d and 2d distance

-- Same example as 3D closest point example

SELECT ST\_3DDistance(poly, mline) As dist3d,

ST\_Distance(poly, mline) As dist2d

FROM (SELECT

ST\_GeomFromEWKT(’POLYGON((175 150 5, 20 40 5, 35 45 5, 50 60 5, 100

←

100 5, 175 150 5))’) As poly,

ST\_GeomFromEWKT(’MULTILINESTRING((175 155 2, 20 40 20, 50 60 -2, 125 100 1, ←

175 155 1),

(1 10 2, 5 20 1))’) As mline ) As foo;

dist3d

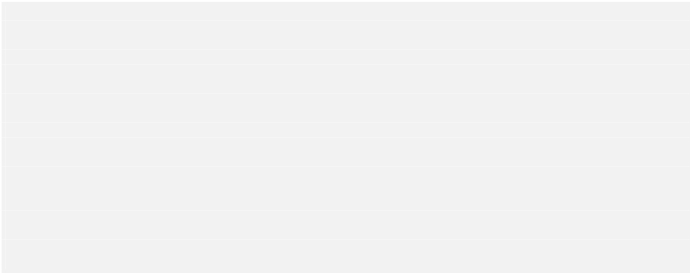
| dist2d

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

0.716635696066337 | 0

**请参考**

ST\_Distance, ST\_3DClosestPoint, ST\_3DDWithin, ST\_3DMaxDistance, ST\_3DShortestLine, ST\_Transform



PostGIS 2.2.0dev 开发手册

233 / 686

**8.9.3**

**ST\_3DDWithin**

ST\_3DDWithin —如果两个几何对象的三维距离指定的距离值范围内，则返回TRUE

**用法**

boolean ST\_3DDWithin(geometry g1, geometry g2, double precision distance\_of\_srid);

**描述**

如果两个几何对象的三维距离指定的距离值范围内，则返回TRUE，距离单位以（SRS表中单位为准）

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

该函数支持 Polyhedral Surface类型几何对象.

该函数方法实现了规范 SQL/MM speciﬁcation. SQL-MM ?

可用版本: 2.0.0

**样例**

-- Geometry example - units in meters (SRID: 2163 US National Atlas Equal area) (3D

and line compared 2D point and line)

point

←

-- 注意: currently no vertical datum support so Z is not transformed and assumed to

units as final.

SELECT ST\_3DDWithin(

ST\_Transform(ST\_GeomFromEWKT(’SRID=4326;POINT(-72.1235 42.3521 4)’),2163),

ST\_Transform(ST\_GeomFromEWKT(’SRID=4326;LINESTRING(-72.1260 42.45 15, -72.123

20)’),2163),

126.8

) As within\_dist\_3d,

ST\_DWithin(

ST\_Transform(ST\_GeomFromEWKT(’SRID=4326;POINT(-72.1235 42.3521 4)’),2163),

ST\_Transform(ST\_GeomFromEWKT(’SRID=4326;LINESTRING(-72.1260 42.45 15, -72.123

20)’),2163),

126.8

) As within\_dist\_2d;

within\_dist\_3d | within\_dist\_2d

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

be same ←

42.1546 ←

42.1546 ←

f

| t

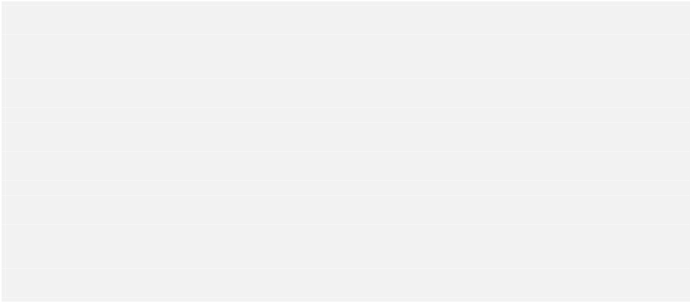
**请参考**

ST\_3DDistance, ST\_Distance, ST\_DWithin, ST\_3DMaxDistance, ST\_Transform

**8.9.4**

**ST\_3DDFullyWithin**

ST\_3DDFullyWithin —如果3D对象的距离在另一个指定的距离值范围内，则返回TRUE



PostGIS 2.2.0dev 开发手册

234 / 686

**用法**

boolean ST\_3DDFullyWithin(geometry g1, geometry g2, double precision distance);

**描述**

如果3D对象的距离在另一个指定的距离值范围内，则返回TRUE。指定的距离单位是SRS表中规定的单位。该函数的输入几何对象必须都在同一个坐标投影中，有相同的SRID

**注意**

该函数使用时候会自动生成一个bounding box，以便充分利用几何对象上的任意一个索引

可用版本: 2.0.0

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

该函数支持 Polyhedral Surface类型几何对象.

**样例**

-- This compares the difference between fully within and distance within as well

-- as the distance fully within for the 2D footprint of the line/point vs. the 3d fully ←

within

SELECT ST\_3DDFullyWithin(geom\_a, geom\_b, 10) as D3DFullyWithin10, ST\_3DDWithin(geom\_a,

geom\_b, 10) as D3DWithin10,

ST\_DFullyWithin(geom\_a, geom\_b, 20) as D2DFullyWithin20,

ST\_3DDFullyWithin(geom\_a, geom\_b, 20) as D3DFullyWithin20 from

(select ST\_GeomFromEWKT(’POINT(1 1 2)’) as geom\_a,

ST\_GeomFromEWKT(’LINESTRING(1 5 2, 2 7 20, 1 9 100, 14 12 3)’) as geom\_b) t1;

d3dfullywithin10 | d3dwithin10 | d2dfullywithin20 | d3dfullywithin20

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

←

f

| t

| t

| f

**请参考**

ST\_3DMaxDistance, ST\_3DDWithin, ST\_DWithin, ST\_DFullyWithin

**8.9.5**

**ST\_3DIntersects**

ST\_3DIntersects —如果几何对象在3维空间内相交，则返回TRUE，该函数只适用于Point和LINESTRING类型

**用法**

boolean ST\_3DIntersects( geometry geomA , geometry geomB );



PostGIS 2.2.0dev 开发手册

235 / 686

**描述**

覆盖、接触（相切）、包含都属于空间相交。上述的任意一种都返回TRUE，那么这些几何对象也是空间相交的。空间不相交则返回FALSE

可用版本: 2.0.0

**注意**

该函数会自动生成一个bounding box用于利用几何对象上的索引

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

该函数支持 Polyhedral Surface类型几何对象.

该函数方法实现了规范 SQL/MM speciﬁcation. SQL-MM 3: ?

**Geometry 样例**

SELECT ST\_3DIntersects(pt, line), ST\_Intersects(pt,line)

FROM (SELECT ’POINT(0 0 2)’::geometry As pt,

’LINESTRING (0 0 1, 0 2 3 )’::geometry As line) As foo;

st\_3dintersects | st\_intersects

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

f

(1 row)

**请参考**

ST\_Intersects

| t

**8.9.6**

**ST\_3DLongestLine**

ST\_3DLongestLine — 返回两个几何对象之间最长的3维线段

**用法**

geometry ST\_3DLongestLine(geometry g1, geometry g2);

**描述**

返回两个几何对象之间的3维最长距离。如果有多个线段都是最长线段，那么返回第一个最长的。该线段的起点在g1，终点在g2.该函数返回的线段的长度与用函数ST\_3DMaxDistance计算g1和g2的距离值一样。

可用版本: 2.0.0

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

该函数支持 Polyhedral Surface类型几何对象.



PostGIS 2.2.0dev 开发手册

236 / 686

**样例**

PostGIS 2.2.0dev 开发手册

237 / 686

linestring and point -- both 3d and 2d longest line

SELECT ST\_AsEWKT(ST\_3DLongestLine(line,pt)) AS lol3d\_line\_pt,

ST\_AsEWKT(ST\_LongestLine(line,pt)) As lol2d\_line\_pt

FROM (SELECT ’POINT(100 100 30)’::geometry As pt,

’LINESTRING (20 80 20, 98 190 1, 110 180 3, 50 75 1000)’:: ←

geometry As line

) As foo;

lol3d\_line\_pt

|

lol2d\_line\_pt

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

LINESTRING(50 75 1000,100 100 30) | LINESTRING(98 190,100 100)

linestring and multipoint -- both 3d and 2d longest line

SELECT ST\_AsEWKT(ST\_3DLongestLine(line,pt)) AS lol3d\_line\_pt,

ST\_AsEWKT(ST\_LongestLine(line,pt)) As lol2d\_line\_pt

FROM (SELECT ’MULTIPOINT(100 100 30, 50 74 1000)’::geometry As pt,

’LINESTRING (20 80 20, 98 190 1, 110 180 3, 50 75 900)’:: ←

geometry As line

) As foo;

lol3d\_line\_pt

|

lol2d\_line\_pt

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

LINESTRING(98 190 1,50 74 1000) | LINESTRING(98 190,50 74)

Multilinestring and polygon both 3d and 2d longest line

SELECT ST\_AsEWKT(ST\_3DLongestLine(poly, mline)) As lol3d,

ST\_AsEWKT(ST\_LongestLine(poly, mline)) As lol2d

FROM (SELECT

ST\_GeomFromEWKT(’POLYGON((175 150 5, 20 40 5, 35 45 5, 50 60 5,

←

100 100 5, 175 150 5))’) As poly,

ST\_GeomFromEWKT(’MULTILINESTRING((175 155 2, 20 40 20, 50 60 -2, 125

100 1, 175 155 1),

(1 10 2, 5 20 1))’) As mline ) As foo;

←

lol3d

|

lol2d

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

LINESTRING(175 150 5,1 10 2) | LINESTRING(175 150,1 10)

**请参考**

ST\_3DClosestPoint, ST\_3DDistance, ST\_LongestLine, ST\_3DShortestLine, ST\_3DMaxDistance

**8.9.7**

**ST\_3DMaxDistance**

ST\_3DMaxDistance —对于3维的geometry类型对象，返回两个几何对象的最大笛卡尔距离(基于SRS表)，单位以投影系单位为准

**用法**

ﬂoat ST\_3DMaxDistance(geometry g1, geometry g2);

PostGIS 2.2.0dev 开发手册

238 / 686

**描述**

对于3维的geometry类型对象，返回两个几何对象的最大笛卡尔距离(基于SRS表)，单位以投影系单位为准

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

该函数支持 Polyhedral Surface类型几何对象.

可用版本: 2.0.0

**样例**

-- Geometry example - units in meters (SRID: 2163 US National Atlas Equal area) (3D point

←

and line compared 2D point and line)

-- 注意: currently no vertical datum support so Z is not transformed and assumed to be same ←

units as final.

SELECT ST\_3DMaxDistance(

ST\_Transform(ST\_GeomFromEWKT(’SRID=4326;POINT(-72.1235 42.3521 10000)’),2163),

ST\_Transform(ST\_GeomFromEWKT(’SRID=4326;LINESTRING(-72.1260 42.45 15, -72.123 42.1546 ←

20)’),2163)

) As dist\_3d,

ST\_MaxDistance(

ST\_Transform(ST\_GeomFromEWKT(’SRID=4326;POINT(-72.1235 42.3521 10000)’),2163),

ST\_Transform(ST\_GeomFromEWKT(’SRID=4326;LINESTRING(-72.1260 42.45 15, -72.123 42.1546 ←

20)’),2163)

) As dist\_2d;

dist\_3d

|

dist\_2d

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

24383.7467488441 | 22247.8472107251

**请参考**

ST\_Distance, ST\_3DDWithin, ST\_3DMaxDistance, ST\_Transform

**8.9.8**

**ST\_3DShortestLine**

ST\_3DShortestLine —返回两个3维几何对象之间最短的线段

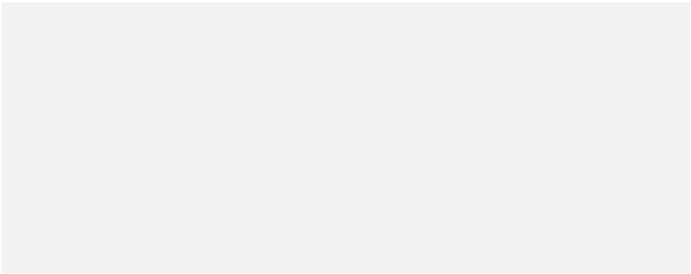
**用法**

geometry ST\_3DShortestLine(geometry g1, geometry g2);

**描述**

返回两个3维几何对象之间最短的线段，如果函数找到这两个3维几何对象之间不止一条线段，那么只会返回第一条。如果g1和g2只在一个点相交，该函数会返回一条返回起点和终点都在该交点的线段。如果g1和g2相交不止一个点，该函数会返回起点和终点都在同一点的线段，但该交点可以是g1和g2相交的任意一点。返回的线段起点总是属于g1，终点属于g2。该函数返回的线段的3D长度总是和ST\_3Ddistance计算g1和g2之间的距离值是一样的。

可用版本: 2.0.0



PostGIS 2.2.0dev 开发手册

239 / 686

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

该函数支持 Polyhedral Surface类型几何对象.

**样例**

linestring and point -- both 3d and 2d shortest line

SELECT ST\_AsEWKT(ST\_3DShortestLine(line,pt)) AS shl3d\_line\_pt,

ST\_AsEWKT(ST\_ShortestLine(line,pt)) As shl2d\_line\_pt

FROM (SELECT ’POINT(100 100 30)’::geometry As pt,

’LINESTRING (20 80 20, 98 190 1, 110 180 3, 50 75 1000)’:: ←

geometry As line

) As foo;

shl3d\_line\_pt

|

←

shl2d\_line\_pt

←

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

LINESTRING(54.6993798867619 128.935022917228 11.5475869506606,100 100 30)

|

←

LINESTRING(73.0769230769231 115.384615384615,100 100)

linestring and multipoint -- both 3d and 2d shortest line

SELECT ST\_AsEWKT(ST\_3DShortestLine(line,pt)) AS shl3d\_line\_pt,

ST\_AsEWKT(ST\_ShortestLine(line,pt)) As shl2d\_line\_pt

FROM (SELECT ’MULTIPOINT(100 100 30, 50 74 1000)’::geometry As pt,

’LINESTRING (20 80 20, 98 190 1, 110 180 3, 50 75 900)’:: ←

geometry As line

) As foo;

shl2d\_line\_pt

shl3d\_line\_pt

|

←

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

LINESTRING(54.6993798867619 128.935022917228 11.5475869506606,100 100 30) | LINESTRING ←

(50 75,50 74)

Multilinestring and polygon both 3d and 2d shortest line

SELECT ST\_AsEWKT(ST\_3DShortestLine(poly, mline)) As shl3d,

ST\_AsEWKT(ST\_ShortestLine(poly, mline)) As shl2d

FROM (SELECT

ST\_GeomFromEWKT(’POLYGON((175 150 5, 20 40 5, 35 45 5, 50 60 5,

←

100 100 5, 175 150 5))’) As poly,

ST\_GeomFromEWKT(’MULTILINESTRING((175 155 2, 20 40 20, 50 60 -2, 125

100 1, 175 155 1),

(1 10 2, 5 20 1))’) As mline ) As foo;

←

shl3d

←

|

shl2d

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

LINESTRING(39.993580415989 54.1889925532825 5,40.4078575708294 53.6052383805529

5.03423778139177) | LINESTRING(20 40,20 40)

←



PostGIS 2.2.0dev 开发手册

240 / 686

**请参考**

ST\_3DClosestPoint, ST\_3DDistance, ST\_LongestLine, ST\_ShortestLine, ST\_3DMaxDistance

**8.9.9**

**ST\_Area**

ST\_Area —返回输入对象是polygon或者multi-polygon对象的面积。对于geometry对象来说，面积单位是SRID规定的单位，对于geography对象，面积是平方米

**用法**

ﬂoat ST\_Area(geometry g1);

ﬂoat ST\_Area(geography geog, boolean use\_spheroid=true);

**描述**

返回输入对象是polygon或者multi-polygon对象的面积。返回ST\_Surface 或ST\_MultiSurface value的面积值。对于geometry对象来说，面积单位是SRID规定的单位，对于geography对象，面积是平方米。默认计算的是大地坐标系（目前只有WGS84）的下的面积。如果想要计算更快但精度略低，需要计算球面积。使用方法如下：ST\_Area(geog,false).

版本提升: 2.0.0 -开始引入对2D polyhedral surface对象的支持.

该函数方法实现了规范 [OpenGIS Simple Features Implementation Speciﬁcation for SQL 1.1.](http://www.opengeospatial.org/standards/sfs)

该函数方法实现了规范 SQL/MM speciﬁcation. SQL-MM 3: 8.1.2, 9.5.3

该函数支持 Polyhedral Surface类型几何对象.

**注意**

对polyhedral surface对象来说，值支持2D polyhedral surface对象(非 2.5D对象). 对2.5D对象，可能会返回一个非0值，但是只适用于polygon或multipolygon的面完全处于XY平面内的情形

该方法由SFCGAL库提供后台支持.

**样例**

返回马萨诸塞州的一个地块的平方英尺的面积，并转换得到平方米。注意：这里返回的结果是平方英尺，因为SRID为2249，是参考系Mass State Plane 坐标系的单位英尺(译者注：关于该参考系，请参考：http://www.mass.gov/anf/research-and-tech/it-serv-and-support/application-serv/office-of-geographic-information-massgis/datalayers/overview.html)

SELECT ST\_Area(the\_geom) As sqft, ST\_Area(the\_geom)\*POWER(0.3048,2) As sqm

FROM (SELECT

ST\_GeomFromText(’POLYGON((743238 2967416,743238 2967450,

743265 2967450,743265.625 2967416,743238 2967416))’,2249) ) As foo(the\_geom);

sqft

|

sqm

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

928.625 | 86.27208552

Return area square feet and transform to Massachusetts state plane meters (26986) to get square meters. Notice this is in square

feet because 2249 is Mass State Plane Feet and transformed area is in square meters since 26986 is state plane mass meters



PostGIS 2.2.0dev 开发手册

241 / 686

SELECT ST\_Area(the\_geom) As sqft, ST\_Area(ST\_Transform(the\_geom,26986)) As sqm

FROM (SELECT

ST\_GeomFromText(’POLYGON((743238 2967416,743238 2967450,

743265 2967450,743265.625 2967416,743238 2967416))’,2249) ) As foo(the\_geom);

sqft

|

sqm

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

928.625 | 86.2724304199219

Return area square feet and square meters using Geography data type. Notice that we transform to our geometry to geography

(before you can do that make sure your geometry is in WGS 84 long lat 4326). Geography always measures in meters. This is

just for demonstration to compare. Normally your table will be stored in geography data type already.

SELECT ST\_Area(the\_geog)/POWER(0.3048,2) As sqft\_spheroid,

ST\_Area(the\_geog,false)/POWER ←

(0.3048,2) As sqft\_sphere, ST\_Area(the\_geog) As sqm\_spheroid

FROM (SELECT

geography(

ST\_Transform(

ST\_GeomFromText(’POLYGON((743238 2967416,743238 2967450,743265 2967450,743265.625

2967416,743238 2967416))’,

2249

) ,4326

)

)

) As foo(the\_geog);

←

sqft\_spheroid

|

sqft\_sphere

|

sqm\_spheroid

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

928.684405217197 | 927.186481558724 | 86.2776044452694

--if your data is in geography already

SELECT ST\_Area(the\_geog)/POWER(0.3048,2) As

sqft, ST\_Area(the\_geog) As sqm

FROM somegeogtable;

**请参考**

ST\_GeomFromText, ST\_GeographyFromText, ST\_SetSRID, ST\_Transform

**8.9.10**

**ST\_Azimuth**

ST\_Azimuth —返回几何对象点A从正北方向按顺时针旋转到点B时候所需要的角度

**用法**

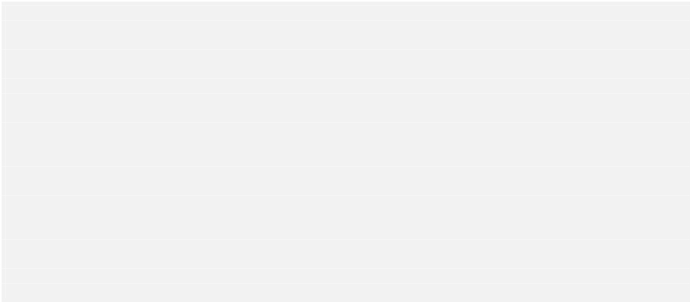
ﬂoat ST\_Azimuth(geometry pointA, geometry pointB);

ﬂoat ST\_Azimuth(geography pointA, geography pointB);

**描述**

返回几何对象点A从正北方向按顺时针旋转到点B时候所需要的角度，如果两个点重合，则返回NULL值。该方位角以正北方向为标准起始线，该线定义如下：North = 0; East = PI/2; South = PI; West = 3PI/2。

方位角是一个定义成角度的数学概念，在这种情形下需要一个参考平面和一个点来定义弧度。



PostGIS 2.2.0dev 开发手册

242 / 686

可用版本: 1.1.0

版本提升: 2.0.0引入支持geography对象.

方位角在与函数ST\_Translate 结合起来使用的时候特别有用，可以用于绕着一个几何对象的垂直轴来旋转一定角度。参考upgis\_lineshift Plpgsqlfunctions PostGIS wiki section获取样例

**样例**

Geometry Azimuth in degrees

SELECT ST\_Azimuth(ST\_Point(25,45), ST\_Point(75,100))/(2\*pi())\*360 as degA\_B,

ST\_Azimuth(ST\_Point(75,100), ST\_Point(25,45))/(2\*pi())\*360 As degB\_A;

-- NOTE easier to remember syntax using PostgreSQL built-in degrees function --

-- Both yield same answer --

SELECT degrees( ST\_Azimuth(ST\_Point(25,45), ST\_Point(75,100)) ) as degA\_B,

degrees( ST\_Azimuth(ST\_Point(75,100), ST\_Point(25,45)) ) As degB\_A;

dega\_b

|

degb\_a

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

42.2736890060937 | 222.273689006094

Green: the start Point(25,45) with its vertical. Yellow:

degA\_B as the path to travel (azimuth).

**请参考**

ST\_Point, ST\_Translate, ST\_Project, [PostgreSQL Math Functions](http://www.postgresql.org/docs/current/interactive/functions-math.html)

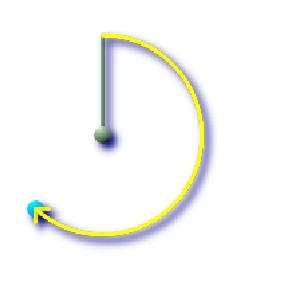
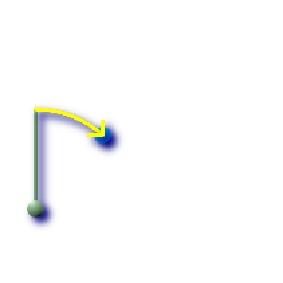
Green: the start Point(75,100) with its vertical. Yellow:

degB\_A as the path to travel (azimuth).

**8.9.11**

**ST\_Centroid**

ST\_Centroid —返回一个几何对象的几何中心.



PostGIS 2.2.0dev 开发手册

243 / 686

**用法**

geometry ST\_Centroid(geometry g1);

**描述**

返回一个几何对象的中心，或者等价地返回一个几何对象的质心，即一个点。关于多点几何对象 [MULTI]POINT，返回的是输入几何对象坐标的的算术平均值。对于[MULTI]LINESTRING几何类型对象，返回的是每个线段的加权长度。对于[MULTI]POLYGONs，加权值以面积来算。如果输入的是一个EMPTY空几何对象，将返回GEOMETRYCOLLECTION类型几何对象。如果输入的是NULL，则返回NULL。几何对象的中心等同于最高维度几何对象几何的中心（因为较低的维度几何对象对中心的加权贡献为0）。

**注意**

如果安装PostGIS添加了GEOS模块，该计算会更精确

该函数方法实现了规范 [OpenGIS Simple Features Implementation Speciﬁcation for SQL 1.1.](http://www.opengeospatial.org/standards/sfs)

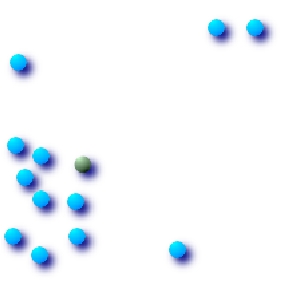
该函数方法实现了规范 SQL/MM speciﬁcation. SQL-MM 3: 8.1.4, 9.5.5

**样例**

在每一个下面的插图中，蓝点代表了输入几何对象的中心

Centroid of a MULTIPOINT

Centroid of a LINESTRING



PostGIS 2.2.0dev 开发手册

244 / 686

Centroid of a POLYGON

Centroid of a GEOMETRYCOLLECTION

SELECT ST\_AsText(ST\_Centroid(’MULTIPOINT ( -1 0, -1 2, -1 3, -1 4, -1 7, 0 1, 0 3, 1 1, 2

0, 6 0, 7 8, 9 8, 10 6 )’));

st\_astext

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

POINT(2.30769230769231 3.30769230769231)

(1 row)

**请参考**

ST\_PointOnSurface

←

**8.9.12**

**ST\_ClosestPoint**

ST\_ClosestPoint — 返回2维平面内离g2几何对象最近的g1上的点。这个点也是g1和g2之间最短的线的第一个点

**用法**

geometry ST\_ClosestPoint(geometry g1, geometry g2);

**描述**

返回2维平面内离g2几何对象最近的g1上的点。这个点也是g1和g2之间最短的线的第一个点

**注意**

如果输入对象是3D几何对象，你应该选择使用ST\_3DClosestPoint.

可用版本: 1.5.0

**样例**



PostGIS 2.2.0dev 开发手册

245 / 686

Closest between point and linestring is the point itself, but

closest point between a linestring and point is the point on

line string that is closest.

closest point on polygon A to polygon B

SELECT ST\_AsText(ST\_ClosestPoint(pt,line) ←

) AS cp\_pt\_line,

ST\_AsText(ST\_ClosestPoint(line,pt ←

)) As cp\_line\_pt

FROM (SELECT ’POINT(100 100)’::geometry ←

As pt,

’LINESTRING (20 80, 98 ←

190, 110 180, 50 75 )’::geometry As line

) As foo;

SELECT ST\_AsText(

ST\_ClosestPoint(

ST\_GeomFromText(’ ←

POLYGON((175 150, 20 40, 50 60, 125 100, 175 150

ST\_Buffer( ←

ST\_GeomFromText(’POINT(110 170)’), 20)

)

) As ptwkt;

ptwkt

cp\_pt\_line

|

←

------------------------------------------ ←

cp\_line\_pt

POINT(140.752120669087 125.695053378061)

POINT(100 100) | POINT(73.0769230769231

←

115.384615384615)

**请参考**

ST\_3DClosestPoint,ST\_Distance, ST\_LongestLine, ST\_ShortestLine, ST\_MaxDistance

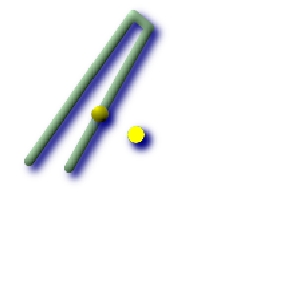
**8.9.13**

**ST\_Contains**

ST\_Contains —当且仅当几何对象B的所有点没有在几何对象A的外部，并且B最少有一个点在A的内部（译者注：意思A把B完全包含了）

**用法**

boolean ST\_Contains(geometry geomA, geometry geomB);



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

PostGIS 2.2.0dev 开发手册

246 / 686

**描述**

当且仅当几何对象B的所有点没有在几何对象A的外部，并且B最少有一个点在A的内部（译者注：意思A把B完全包含了）。该定义的一个重要差别是几何对象不包含B的边界，但A会包含自己的边界。和函数ST\_ContainsProperly对比而言，ST\_ContainsProperly函数几何对象不会完全包含自身边界(译者注：文字翻译如此，待定)。如果几何对象B完全在几何对象A范围内，则返回TRUE。该函数只有在输入的几何对象处于相同的坐标投影系中，有相同的SRID才有使用意义。函数ST\_Contains 是函数ST\_Within逆功能函数. 因此ST\_Contains(A,B)的值和ST\_Within(B,A)值是一样的，除非输入的几何对象是无效的几何对象，因为这种情况下返回的总是错误的。

该函数由GEOS模块支持

**重要**

该函数不支持GEOMETRYCOLLECTION类型对象作为输入参数

**重要**

不要将该函数用于无效的几何对象，否则你会得到不可预料的结果

该函数会自动生成一个bounding box用于利用几何对象上的索引。如果不想使用索引，请使用函数\_ST\_Contains.

注意: 这是一个允许返回逻辑值，而不是整型值的版本

该函数方法实现了规范 OpenGIS Simple Features Implementation Speciﬁcation for SQL 1.1. s2.1.1.2 // s2.1.13.3 -

same as within(geometry B, geometry A)

该函数方法实现了规范 SQL/MM speciﬁcation. SQL-MM 3: 5.1.31

函数ST\_Contains 和 ST\_Within 有一些不直观但很确定的差别。更多细节请查看[Subtleties ofOGC Covers, Contains, Within](http://lin-ear-th-inking.blogspot.com/2007/06/subtleties-of-ogc-covers-spatial.html)

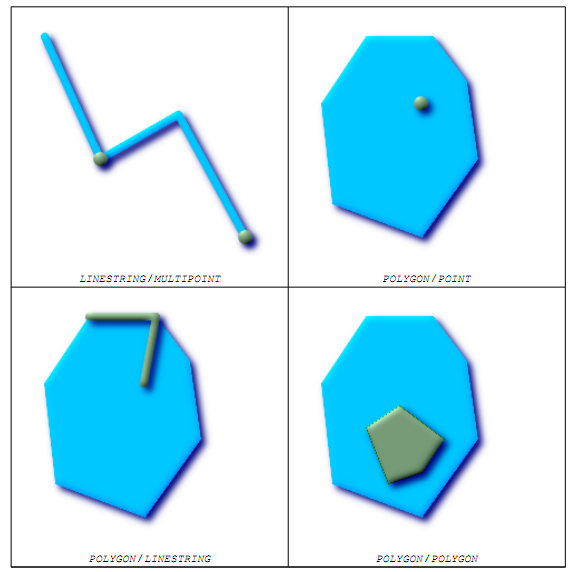
**样例**

函数 ST\_Contains 在如下表格列出的场景中必定会返回TRUE



PostGIS 2.2.0dev 开发手册

247 / 686



ST\_Contains空间谓词在如下情况中返回FALSE

PostGIS 2.2.0dev 开发手册

248 / 686

POLYGON / MULTIPOINT

POLYGON / LINESTRING

-- A circle within a circle

SELECT ST\_Contains(smallc, bigc) As smallcontainsbig,

ST\_Contains(bigc,smallc) As bigcontainssmall,

ST\_Contains(bigc, ST\_Union(smallc, bigc)) as bigcontainsunion,

ST\_Equals(bigc, ST\_Union(smallc, bigc)) as bigisunion,

ST\_Covers(bigc, ST\_ExteriorRing(bigc)) As bigcoversexterior,

ST\_Contains(bigc, ST\_ExteriorRing(bigc)) As bigcontainsexterior

FROM (SELECT ST\_Buffer(ST\_GeomFromText(’POINT(1 2)’), 10) As smallc,

ST\_Buffer(ST\_GeomFromText(’POINT(1 2)’), 20) As bigc) As foo;

-- Result

smallcontainsbig | bigcontainssmall | bigcontainsunion | bigisunion | bigcoversexterior | ←

bigcontainsexterior

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

f

| t

| t

| t

| t

| f

-- Example demonstrating difference between contains and contains properly

SELECT ST\_GeometryType(geomA) As geomtype, ST\_Contains(geomA,geomA) AS acontainsa,

ST\_ContainsProperly(geomA, geomA) AS acontainspropa,

ST\_Contains(geomA, ST\_Boundary(geomA)) As acontainsba, ST\_ContainsProperly(geomA,

ST\_Boundary(geomA)) As acontainspropba

FROM (VALUES ( ST\_Buffer(ST\_Point(1,1), 5,1) ),

( ST\_MakeLine(ST\_Point(1,1), ST\_Point(-1,-1) ) ),

( ST\_Point(1,1) )

) As foo(geomA);

←

←

geomtype

| acontainsa | acontainspropa | acontainsba | acontainspropba

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

ST\_Polygon

| t

| f

| f

| f

ST\_LineString | t

| f

| f

| f

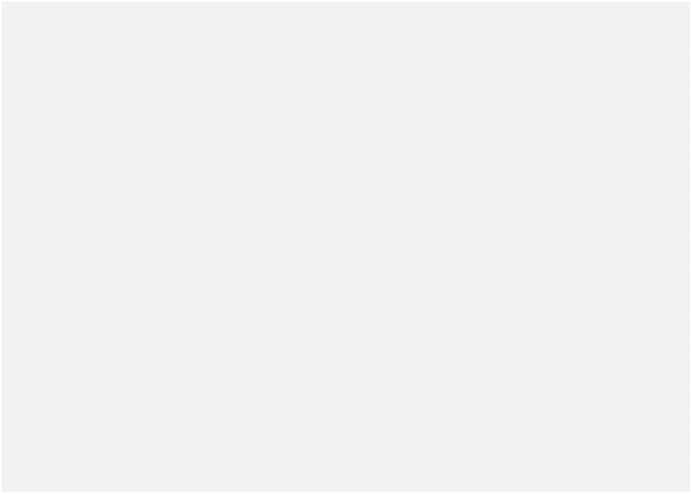
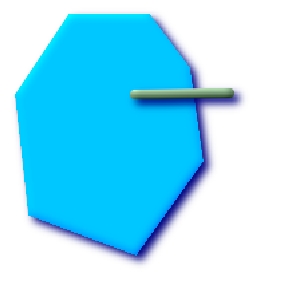
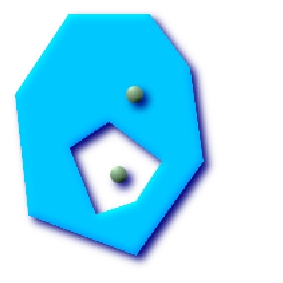
ST\_Point

| t

| t

| f

| f



PostGIS 2.2.0dev 开发手册

249 / 686

**请参考**

ST\_Boundary, ST\_ContainsProperly, ST\_Covers, ST\_CoveredBy, ST\_Equals, ST\_Within

**8.9.14**

**ST\_ContainsProperly**

ST\_ContainsProperly — 如果输入几何对象B和几何对象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(geometry geomA, geometry geomB);

**描述**

如果输入几何对象B和几何对象A的内部相交，但不和A的边界（或外部）有接触，那么返回TRUE。每个其他几何对象的点是该几何对象内部的一个点。对于DE-9IM 规范的两个几何对象的的相交结果矩阵与ST\_Relate函数使用的[T\*\*FF\*FF\*]相匹配。

**注意**

从JTS（译者注：著名的空间拓扑算法JAVA API实现，符合OGC相关规范）文档轻微改动一下：该函数相比较函数ST\_Contains 和函数 ST\_Intersects 的优势是该函数计算更高效，因为没有必要对每个点计算拓扑的必要。一个使用该函数的例子是计算包含一个很大的多边形几何对象的集合的交集。因为计算交集是一个相当慢的操作，使用函数ST\_ContainsProperly来过滤掉完全在区域内部的测试对象会更高效。在这些样例中，交集是一个优先的原始测试几何对象

可用版本: 1.4.0 -需要GEOS版本>= 3.1.0.

**重要**

该函数不支持GEOMETRYCOLLECTION类型对象作为输入参数

**重要**

不要将该函数应用于无效的几何对象上，否则你会得到不可预料的结果.

该函数使用时候会自动生成一个bounding box，以便充分利用几何对象上的任意一个索引. 如果想避免使用索引，使用函数\_ST\_ContainsProperly.

**样例**



PostGIS 2.2.0dev 开发手册

250 / 686

--a circle within a circle

SELECT ST\_ContainsProperly(smallc, bigc) As smallcontainspropbig,

ST\_ContainsProperly(bigc,smallc) As bigcontainspropsmall,

ST\_ContainsProperly(bigc, ST\_Union(smallc, bigc)) as bigcontainspropunion,

ST\_Equals(bigc, ST\_Union(smallc, bigc)) as bigisunion,

ST\_Covers(bigc, ST\_ExteriorRing(bigc)) As bigcoversexterior,

ST\_ContainsProperly(bigc, ST\_ExteriorRing(bigc)) As bigcontainsexterior

FROM (SELECT ST\_Buffer(ST\_GeomFromText(’POINT(1 2)’), 10) As smallc,

ST\_Buffer(ST\_GeomFromText(’POINT(1 2)’), 20) As bigc) As foo;

--Result

smallcontainspropbig | bigcontainspropsmall | bigcontainspropunion | bigisunion |

bigcoversexterior | bigcontainsexterior

←

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

f

| t

| f

| t

| t

←

| f

--example demonstrating difference between contains and contains properly

SELECT ST\_GeometryType(geomA) As geomtype, ST\_Contains(geomA,geomA) AS acontainsa,

ST\_ContainsProperly(geomA, geomA) AS acontainspropa,

ST\_Contains(geomA, ST\_Boundary(geomA)) As acontainsba, ST\_ContainsProperly(geomA,

ST\_Boundary(geomA)) As acontainspropba

FROM (VALUES ( ST\_Buffer(ST\_Point(1,1), 5,1) ),

( ST\_MakeLine(ST\_Point(1,1), ST\_Point(-1,-1) ) ),

( ST\_Point(1,1) )

) As foo(geomA);

←

←

geomtype

| acontainsa | acontainspropa | acontainsba | acontainspropba

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

ST\_Polygon

| t

| f

| f

| f

ST\_LineString | t

| f

| f

| f

ST\_Point

| t

| t

| f

| f

**请参考**

ST\_GeometryType, ST\_Boundary, ST\_Contains, ST\_Covers, ST\_CoveredBy, ST\_Equals, ST\_Relate, ST\_Within

**8.9.15**

**ST\_Covers**

ST\_Covers —如果geometry或geography对象B的所有点都不在geometry或geography对象A的外部，则返回1（即TRUE）

**用法**

boolean ST\_Covers(geometry geomA, geometry geomB);

boolean ST\_Covers(geography geogpolyA, geography geogpointB);

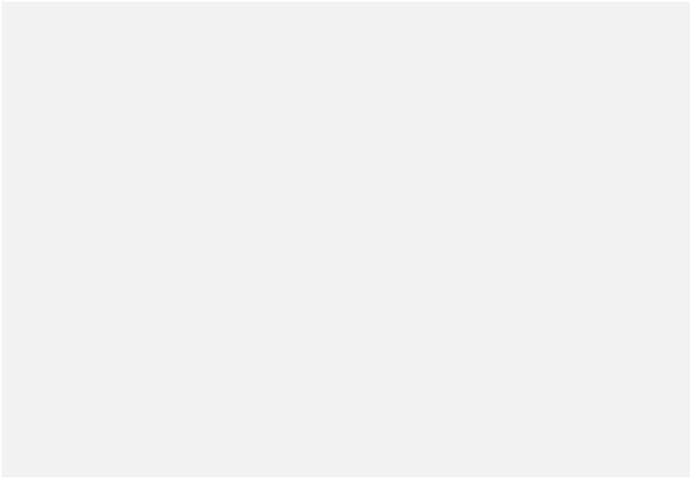
**描述**

如果geometry或geography对象B的所有点都不在geometry或geography对象A的外部，则返回1（即TRUE）

该函数由GEOS模块支持

**重要**

该函数不支持GEOMETRYCOLLECTION类型对象作为输入参数



PostGIS 2.2.0dev 开发手册

251 / 686

**重要**

对于geography类型，只支持Polygon覆盖point这样的场景.

**重要**

不要将该函数应用于无效的几何对象上，否则你会得到不可预料的结果.

该函数使用时候会自动生成一个bounding box，以便充分利用几何对象上的任意一个索引。如果想避免使用索引，使用函数\_ST\_Covers.

可用版本: 1.2.2 -需要GEOS版本>= 3.0

可用版本: 1.5 -引入支持geography对象.

这是一个返回逻辑值而不是整型值的可用版本.

这个函数非OGC标准函数，oracle也有这个函数

该函数与函数ST\_Contains 和 ST\_Within有不显而易见但却真实存在的细微差别。详细情况请查看 [Subtleties of](http://lin-ear-th-inking.blogspot.com/2007/06/subtleties-of-ogc-covers-spatial.html) [OGC Covers, Contains, Within](http://lin-ear-th-inking.blogspot.com/2007/06/subtleties-of-ogc-covers-spatial.html)

**样例**

Geometry example

--a circle covering a circle

SELECT ST\_Covers(smallc,smallc) As smallinsmall,

ST\_Covers(smallc, bigc) As smallcoversbig,

ST\_Covers(bigc, ST\_ExteriorRing(bigc)) As bigcoversexterior,

ST\_Contains(bigc, ST\_ExteriorRing(bigc)) As bigcontainsexterior

FROM (SELECT ST\_Buffer(ST\_GeomFromText(’POINT(1 2)’), 10) As smallc,

ST\_Buffer(ST\_GeomFromText(’POINT(1 2)’), 20) As bigc) As foo;

--Result

smallinsmall | smallcoversbig | bigcoversexterior | bigcontainsexterior

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

t

| f

| t

| f

(1 row)

Geeography Example

-- a point with a 300 meter buffer compared to a point, a point and its 10 meter buffer

SELECT ST\_Covers(geog\_poly, geog\_pt) As poly\_covers\_pt,

ST\_Covers(ST\_Buffer(geog\_pt,10), geog\_pt) As buff\_10m\_covers\_cent

FROM (SELECT ST\_Buffer(ST\_GeogFromText(’SRID=4326;POINT(-99.327 31.4821)’), 300) As

geog\_poly,

ST\_GeogFromText(’SRID=4326;POINT(-99.33 31.483)’) As geog\_pt ) As foo;

poly\_covers\_pt | buff\_10m\_covers\_cent

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

←

f

| t

**请参考**

ST\_Contains, ST\_CoveredBy, ST\_Within



PostGIS 2.2.0dev 开发手册

252 / 686

**8.9.16**

**ST\_CoveredBy**

ST\_CoveredBy —如果geometry或geography对象A的所有点都不在geometry或geography对象B的外部，则返回1（即TRUE）

**用法**

boolean ST\_CoveredBy(geometry geomA, geometry geomB);

boolean ST\_CoveredBy(geography geogA, geography geogB);

**描述**

如果geometry或geography对象A的所有点都不在geometry或geography对象B的外部，则返回1（即TRUE）

该函数由GEOS模块支持

**重要**

该函数不支持GEOMETRYCOLLECTION类型对象作为输入参数

**重要**

不要将该函数应用于无效的几何对象上，否则你会得到不可预料的结果.

可用版本: 1.2.2 –需要 GEOS >= 3.0

该函数使用时候会自动生成一个bounding box，以便充分利用几何对象上的任意一个索引。如果想避免使用索引，请使用函数\_ST\_CoveredBy.

这是一个返回逻辑值而不是整型值的可用版本.

这个函数非OGC标准函数，oracle也有这个函数

该函数与函数ST\_Contains 和 ST\_Within有不显而易见但却真实存在的细微差别。

详细情况请查看 [Subtleties of](http://lin-ear-th-inking.blogspot.com/2007/06/subtleties-of-ogc-covers-spatial.html) [OGC Covers, Contains, Within](http://lin-ear-th-inking.blogspot.com/2007/06/subtleties-of-ogc-covers-spatial.html)

**样例**

--a circle coveredby a circle

SELECT ST\_CoveredBy(smallc,smallc) As smallinsmall,

ST\_CoveredBy(smallc, bigc) As smallcoveredbybig,

ST\_CoveredBy(ST\_ExteriorRing(bigc), bigc) As exteriorcoveredbybig,

ST\_Within(ST\_ExteriorRing(bigc),bigc) As exeriorwithinbig

FROM (SELECT ST\_Buffer(ST\_GeomFromText(’POINT(1 2)’), 10) As smallc,

ST\_Buffer(ST\_GeomFromText(’POINT(1 2)’), 20) As bigc) As foo;

--Result

smallinsmall | smallcoveredbybig | exteriorcoveredbybig | exeriorwithinbig

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

t

| t

| t

| f

(1 row)

**请参考**

ST\_Contains, ST\_Covers, ST\_ExteriorRing, ST\_Within



PostGIS 2.2.0dev 开发手册

253 / 686

**8.9.17**

**ST\_Crosses**

ST\_Crosses —如果两个输入的对象有部分交集但不是完全相交，则该函数返回TRUE

**用法**

boolean ST\_Crosses(geometry g1, geometry g2);

**描述**

如果两个输入的对象有部分交集但不是完全相交，则该函数返回TRUE.输入的几何对象之间的交集不能是空集，同时相交集合对象的维度必须比两个输入对象的最大维度值要小。除此之外，两个几何对象相交的部分不能是输入的其中任意一个对象（译者注：这种即一个对象包含了另一个对象）。否则返回FALSE。上面的描述，用下面的数学公式表达如下

两个几何对象的DE-9IM 相交矩阵可能结果如下：

· T\*T\*\*\*\*\*\* (对你 Point/Line, Point/Area, 和 Line/Area 情形)

· T\*\*\*\*\*T\*\* (对 Line/Point, Area/Point, 和 Area/Line 情形)

· 0\*\*\*\*\*\*\*\* (对 Line/Line 情形)

对于任意其他维度的组合情况，返回值都是false

OpenGIS的 Simple Features Speciﬁcation 规范定义了该函数只适用于Point/Line, Point/Area, Line/Line, 和 Line/Area情况。JTS 或GEOS扩展了这个规范的定义来对Line/Point, Area/Point and Area/Line 一样支持。这让两者关系看起来对称一些。（译者注：这里的对称就是Point/Line和Line/Point看起来对称）

**重要**

该函数不支持GEOMETRYCOLLECTION类型对象作为输入参数

**注意**

该函数会自动生成一个bounding box以便充分利用几何对象上的任意索引.

该函数方法实现了规范 OpenGIS Simple Features Implementation Speciﬁcation for SQL 1.1. s2.1.13.3

该函数方法实现了规范 SQL/MM speciﬁcation. SQL-MM 3: 5.1.29

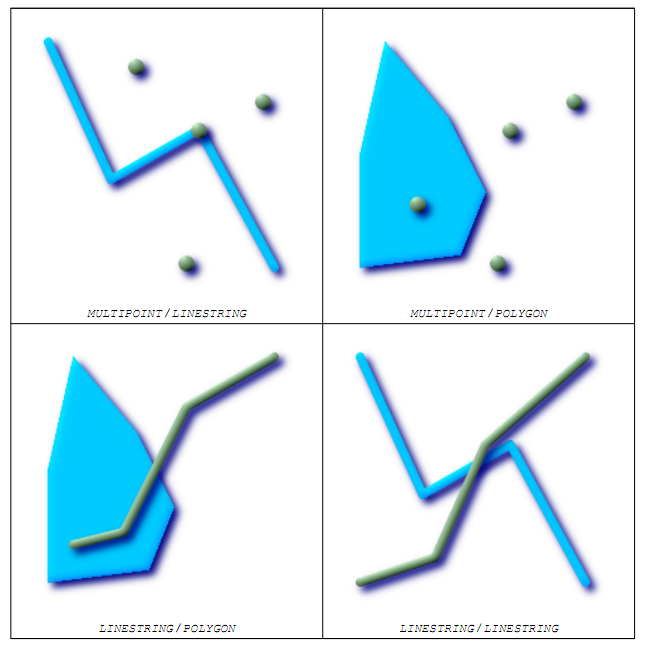


PostGIS 2.2.0dev 开发手册

254 / 686

**样例**

下表列出的均返回TRUE.



考虑一个用户有两个表的场景：一个表名是roads，一个表名是highways

PostGIS 2.2.0dev 开发手册

255 / 686

CREATE TABLE roads (

id serial NOT NULL,

the\_geom geometry,

CONSTRAINT roads\_pkey PRIMARY KEY ( ←

road\_id)

);

CREATE TABLE highways (

id serial NOT NULL,

the\_gem geometry,

CONSTRAINT roads\_pkey PRIMARY KEY ( ←

road\_id)

);

为了得到穿过高速公路的所有普通路的记录，使用类似如下查询

SELECT roads.id

FROM roads, highways

WHERE ST\_Crosses(roads.the\_geom, highways.the\_geom);

**8.9.18**

**ST\_LineCrossingDirection**

ST\_LineCrossingDirection —根据两个LINESTRING，返回一个范围在-3到3之间的数字，该数字表示这两个LINESTRING的相交方式。0表示这两个几何对象没有相交关系

**用法**

integer ST\_LineCrossingDirection(geometry linestringA, geometry linestringB);

**描述**

根据两个LINESTRING，返回一个范围在-3到3之间的数字，该数字表示这两个LINESTRING的相交方式。0表示这两个几何对象没有相交关系。该函数返回的整型常量值定义如下：

· 0: LINE NO CROSS

· -1: LINE CROSS LEFT

· 1: LINE CROSS RIGHT

· -2: LINE MULTICROSS END LEFT

· 2: LINE MULTICROSS END RIGHT

· -3: LINE MULTICROSS END SAME FIRST LEFT

· 3: LINE MULTICROSS END SAME FIRST RIGHT

可用版本: 1.4

**样例**

PostGIS 2.2.0dev 开发手册

256 / 686

Line 1 (green), Line 2 ball is start point, triangle are end

points. Query below.

SELECT ST\_LineCrossingDirection(foo.line1 ←

, foo.line2) As l1\_cross\_l2 ,

ST\_LineCrossingDirection(foo. ←

line2, foo.line1) As l2\_cross\_l1

FROM (

SELECT

Line 1 (green), Line 2 (blue) ball is start point, triangle are

end points. Query below.

SELECT ST\_LineCrossingDirection(foo.line1 ←

, foo.line2) As l1\_cross\_l2 ,

ST\_LineCrossingDirection(foo. ←

line2, foo.line1) As l2\_cross\_l1

FROM (

SELECT

ST\_GeomFromText(’LINESTRING(25 169,89

114,40 70,86 43)’) As line1,

ST\_GeomFromText(’LINESTRING(171 154,20

140,71 74,161 53)’) As line2

) As foo;

l1\_cross\_l2 | l2\_cross\_l1

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

3 | -3

←

←

ST\_GeomFromText(’LINESTRING(25 169,89

114,40 70,86 43)’) As line1,

ST\_GeomFromText(’LINESTRING (171 154,

20 140, 71 74, 2.99 90.16)’) As line2

) As foo;

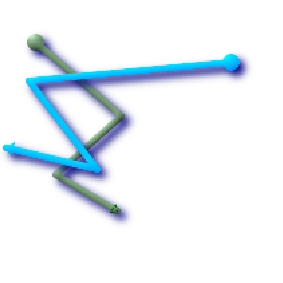
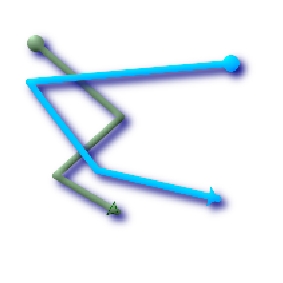
l1\_cross\_l2 | l2\_cross\_l1

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

2 | -2

←

←



PostGIS 2.2.0dev 开发手册

257 / 686

Line 1 (green), Line 2 (blue) ball is start point, triangle are

end points. Query below.

SELECT

ST\_LineCrossingDirection(foo. ←

line1, foo.line2) As l1\_cross\_l2 ,

ST\_LineCrossingDirection(foo. ←

line2, foo.line1) As l2\_cross\_l1

FROM (

Line 1 (green), Line 2 (blue) ball is start point, triangle are

end points. Query below.

SELECT ST\_LineCrossingDirection(foo.line1 ←

, foo.line2) As l1\_cross\_l2 ,

ST\_LineCrossingDirection(foo. ←

line2, foo.line1) As l2\_cross\_l1

FROM (SELECT

SELECT

ST\_GeomFromText(’LINESTRING(25

←

ST\_GeomFromText(’LINESTRING(25 169,89

←

169,89 114,40 70,86 43)’) As line1,

114,40 70,86 43)’) As line1,

ST\_GeomFromText(’LINESTRING (20 140, 71 ←

74, 161 53)’) As line2

ST\_GeomFromText(’LINESTRING(2.99

90.16,71 74,20 140,171 154)’) As line2

) As foo;

←

) As foo;

l1\_cross\_l2 | l2\_cross\_l1

l1\_cross\_l2 | l2\_cross\_l1

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

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

-2 | 2

-1 | 1

SELECT s1.gid, s2.gid, ST\_LineCrossingDirection(s1.the\_geom, s2.the\_geom)

FROM streets s1 CROSS JOIN streets s2 ON (s1.gid != s2.gid AND s1.the\_geom && s2.the\_geom ←

)

WHERE ST\_CrossingDirection(s1.the\_geom, s2.the\_geom) > 0;

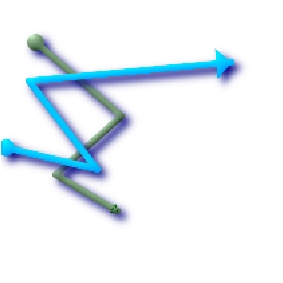
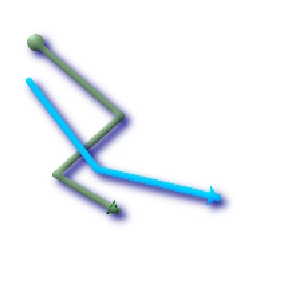
**请参考**

ST\_Crosses

**8.9.19**

**ST\_Disjoint**

ST\_Disjoint —如果两个几何对象没有空间相交则返回TRUE,如果他们没有共同的空间



PostGIS 2.2.0dev 开发手册

258 / 686

**用法**

boolean ST\_Disjoint( geometry A , geometry B );

**描述**

覆盖、相切、包含都不是空间不想交，如果上述的任意一种情况返回TRUE，那么都意味着几何对象没有空间不想交。空间相交的话该函数返回FALSE。

**重要**

该函数不支持GEOMETRYCOLLECTION类型对象作为输入参数

该函数由GEOS模块添加支持

**注意**

该函数调用时候不会使用索引

**注意**

注意: 这是一个“允许”返回boolean值而不是integer值的版本

该函数方法实现了规范 OpenGIS Simple Features Implementation Speciﬁcation for SQL 1.1. s2.1.1.2 //s2.1.13.3 -

a.Relate(b, ’FF\*FF\*\*\*\*’)

该函数方法实现了规范 SQL/MM speciﬁcation. SQL-MM 3: 5.1.26

**样例**

SELECT ST\_Disjoint(’POINT(0 0)’::geometry, ’LINESTRING ( 2 0, 0 2 )’::geometry);

st\_disjoint

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

t

(1 row)

SELECT ST\_Disjoint(’POINT(0 0)’::geometry, ’LINESTRING ( 0 0, 0 2 )’::geometry);

st\_disjoint

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

f

(1 row)

**请参考**

ST\_IntersectsST\_Intersects



PostGIS 2.2.0dev 开发手册

259 / 686

**8.9.20**

**ST\_Distance**

ST\_Distance — 对于geometry类型对象，返回两个几何对象的2维的最小笛卡尔距离。对于geography类型对象，返回WGS84参考系两个geography对象之间的最小空间距离，距离单位是米

**用法**

ﬂoat ST\_Distance(geometry g1, geometry g2);

ﬂoat ST\_Distance(geography gg1, geography gg2);

ﬂoat ST\_Distance(geography gg1, geography gg2, boolean use\_spheroid);

**描述**

对于geometry类型对象，返回两个几何对象的2维的最小笛卡尔距离。对于geography类型对象，返回WGS84参考系两个geography对象之间的最小空间距离，距离单位是米，如果use\_spheroid参数传递的值是false，那么返回的是球面距离，而不是椭圆面距离

该函数方法实现了规范 [OpenGIS Simple Features Implementation Speciﬁcation for SQL 1.1.](http://www.opengeospatial.org/standards/sfs)

该函数方法实现了规范 SQL/MM speciﬁcation. SQL-MM 3: 5.1.23

该函数支持CircularString和Curve几何类型对象

该方法有SFCGAL 库提供支持

可用版本: 1.5.0 1.5版本引入支持geography对象. 提升了2维处理能力以便处理更大更多的顶点几何对象

版本提升: 2.1.0 提高处理geography对象的速度. See Making Geography faster for details.

版本提升: 2.1.0 - 引入对curve几何类型对象的支持.

**基础几何对象样例**

--Geometry example - units in planar degrees 4326 is WGS 84 long lat unit=degrees

SELECT ST\_Distance(

ST\_GeomFromText(’POINT(-72.1235 42.3521)’,4326),

ST\_GeomFromText(’LINESTRING(-72.1260 42.45, -72.123 42.1546)’, 4326)

);

st\_distance

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

0.00150567726382282

-- Geometry example - units in meters (SRID: 26986 Massachusetts state plane meters) (most

accurate for Massachusetts)

SELECT ST\_Distance(

ST\_Transform(ST\_GeomFromText(’POINT(-72.1235 42.3521)’,4326),26986),

ST\_Transform(ST\_GeomFromText(’LINESTRING(-72.1260 42.45, -72.123 42.1546)’, 4326) ←

,26986)

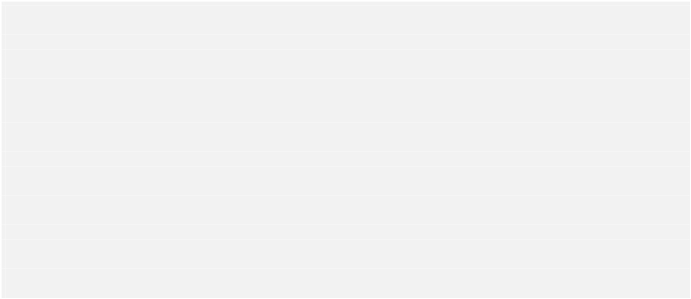
);

st\_distance

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

123.797937878454

←



PostGIS 2.2.0dev 开发手册

260 / 686

-- Geometry example - units in meters (SRID: 2163 US National Atlas Equal area) (least

←

accurate)

SELECT ST\_Distance(

ST\_Transform(ST\_GeomFromText(’POINT(-72.1235 42.3521)’,4326),2163),

ST\_Transform(ST\_GeomFromText(’LINESTRING(-72.1260 42.45, -72.123 42.1546)’, 4326) ←

,2163)

);

st\_distance

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

126.664256056812

**Geography 样例**

-- same as geometry example but note units in meters - use sphere for slightly faster less

accurate

SELECT ST\_Distance(gg1, gg2) As spheroid\_dist, ST\_Distance(gg1, gg2, false) As sphere\_dist

FROM (SELECT

ST\_GeographyFromText(’SRID=4326;POINT(-72.1235 42.3521)’) As gg1,

ST\_GeographyFromText(’SRID=4326;LINESTRING(-72.1260 42.45, -72.123 42.1546)’) As gg2

←

) As foo

;

spheroid\_dist

|

sphere\_dist

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

123.802076746848 | 123.475736916397

**请参考**

ST\_3DDistance, ST\_DWithin, ST\_Distance\_Sphere, ST\_Distance\_Spheroid, ST\_MaxDistance, ST\_Transform

**8.9.21**

**ST\_HausdorffDistance**

ST\_HausdorffDistance — 返回两个几何对象的Hausdorff 距离。该距离用来表示两个几何对象有多类似或有多不相同。距离单位是这两个几何对象所在的空间参考系所用的距离单位

**用法**

ﬂoat ST\_HausdorffDistance(geometry g1, geometry g2);

ﬂoat ST\_HausdorffDistance(geometry g1, geometry g2, ﬂoat densifyFrac);

**描述**

该函数是离散豪斯多夫距离的算法实现。这是针对于离散点的豪斯多夫距离。参考 Wikipedia article on Hausdorff distance [Martin Davis](http://lin-ear-th-inking.blogspot.com/2009/01/computing-geometric-similarity.html) [note on how Hausdorff Distance calculation was used to prove correctness of the CascadePolygonUnion approach.](http://lin-ear-th-inking.blogspot.com/2009/01/computing-geometric-similarity.html)

当指定参数densifyFrac时候，该函数在计算离散豪斯多夫距离前会先把一个分段致密化。

参数 densifyFrac 会将每一个分段致密化。每一个分段会被分割成距离相等的更小分段，而所有分段的总距离最接近给出的大分段总距离。（译者注：这个就是微积分计算的方式啦。。。。）

**注意**

该函数当前的实现只支持离散位置的点。该函数可以允许扩展到任意点的数量。

.



PostGIS 2.2.0dev 开发手册

261 / 686

**注意**

注意该函数实现的算法不等同于标准的豪斯多夫距离。然而对于一个很大的几何对象集合来说，这是一个很接近的结果。该函数一个重要的适用对象是LINESTRING，这些几何对象必须相互之间基本平行，长度基本相等。对于LINESTRING的相似性度量来说很有用。（译者注：一个线段的基本特征就是方向和长度，如果这两者都基本一样，该函数实现的豪斯多夫近似算法会认为这些对象是同一个对象）

可用版本: 1.5.0 -需要GEOS版本>= 3.2.0

**样例**

For each building, ﬁnd the parcel that best represents it. First we require the parcel intersect with the geometry. DISTINCT ON

guarantees we get each building listed only once, the ORDER BY .. ST\_HausdorffDistance gives us a preference of parcel that

is most similar to the building.

SELECT DISTINCT ON(buildings.gid) buildings.gid, parcels.parcel\_id

FROM buildings INNER JOIN parcels ON ST\_Intersects(buildings.geom,parcels.geom)

ORDER BY buildings.gid, ST\_HausdorffDistance(buildings.geom, parcels.geom);

postgis=# SELECT ST\_HausdorffDistance(

’LINESTRING (0 0, 2 0)’::geometry,

’MULTIPOINT (0 1, 1 0, 2 1)’::geometry);

st\_hausdorffdistance

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

1

(1 row)

postgis=# SELECT st\_hausdorffdistance(’LINESTRING (130 0, 0 0, 0 150)’::geometry, ’ ←

LINESTRING (10 10, 10 150, 130 10)’::geometry, 0.5);

st\_hausdorffdistance

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

70

(1 row)

**8.9.22**

**ST\_MaxDistance**

ST\_MaxDistance — 返回两个几何对象的最长距离，距离单位是投影单位

**用法**

ﬂoat ST\_MaxDistance(geometry g1, geometry g2);

**描述**

**注意**

返回两个几何对象的最大距离，单位以投影系的单位为准。如果g1和g2的几何类型相同，该函数会返回两个几何对象上的点的最大距离

可用版本: 1.5.0



PostGIS 2.2.0dev 开发手册

262 / 686

**样例**

Basic furthest distance the point is to any part of the line

postgis=# SELECT ST\_MaxDistance(’POINT(0 0)’::geometry, ’LINESTRING ( 2 0, 0 2 )’::geometry ←

);

st\_maxdistance

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

2

(1 row)

postgis=# SELECT ST\_MaxDistance(’POINT(0 0)’::geometry, ’LINESTRING ( 2 2, 2 2 )’::geometry ←

);

st\_maxdistance

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

2.82842712474619

(1 row)

**请参考**

ST\_Distance, ST\_LongestLine, ST\_DFullyWithin

**8.9.23**

**ST\_Distance\_Sphere**

ST\_Distance\_Sphere — 返回两个以经纬度表示的几何对象的最小距离。该函数使用一个半径为6370986 球体做参照。该函数的计算比ST\_Distance\_Spheroid要快，但是精度要差一些（译者注：因为地球不能完全看做一个正规的球体）。1.5版本之前的PostGIS的该函数只实现了对Point类型对象的支持

**用法**

ﬂoat ST\_Distance\_Sphere(geometry geomlonlatA, geometry geomlonlatB);

**描述**

返回两个以经纬度表示的几何对象的最小距离。该函数使用一个半径为6370986 球体做参照。该函数的计算比ST\_Distance\_Spheroid要快，但是精度要差一些（译者注：因为地球不能完全看做一个正规的球体）。1.5版本之前的PostGIS的该函数只实现了对Point类型对象的支持

**注意**

该函数当前不看几何对象的SRID，因为它会假定给定的几何对象是在WGS 84坐标系中的，坐标用经纬度表示。该函数早前的版本只支持Point

可用版本: 1.5 -支持除了点类型以外的几何对象，早前的版本只支持点类型对象.

**样例**

SELECT round(CAST(ST\_Distance\_Sphere(ST\_Centroid(the\_geom), ST\_GeomFromText(’POINT(-118 38) ←

’,4326)) As numeric),2) As dist\_meters,

round(CAST(ST\_Distance(ST\_Transform(ST\_Centroid(the\_geom),32611),

ST\_Transform(ST\_GeomFromText(’POINT(-118 38)’, 4326),32611)) As numeric),2) As

dist\_utm11\_meters,

←

round(CAST(ST\_Distance(ST\_Centroid(the\_geom), ST\_GeomFromText(’POINT(-118 38)’, 4326)) As

numeric),5) As dist\_degrees,

round(CAST(ST\_Distance(ST\_Transform(the\_geom,32611),

←



PostGIS 2.2.0dev 开发手册

263 / 686

ST\_Transform(ST\_GeomFromText(’POINT(-118 38)’, 4326),32611)) As numeric),2) As

min\_dist\_line\_point\_meters

FROM

←

(SELECT ST\_GeomFromText(’LINESTRING(-118.584 38.374,-118.583 38.5)’, 4326) As the\_geom)

as foo;

dist\_meters | dist\_utm11\_meters | dist\_degrees | min\_dist\_line\_point\_meters

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

70424.47 | 70438.00 | 0.72900 | 65871.18

**请参考**

ST\_Distance, ST\_Distance\_Spheroid

←

**8.9.24**

**ST\_Distance\_Spheroid**

ST\_Distance\_Spheroid —根据给定的椭球面参考系，计算两个经纬度表示的几何对象的最小距离。PostGIS 1.5版本之前只支持Point类型对象

**用法**

ﬂoat ST\_Distance\_Spheroid(geometry geomlonlatA, geometry geomlonlatB, spheroid measurement\_spheroid);

**描述**

根据给定的椭球面参考系，计算两个经纬度表示的几何对象的最小距离。PostGIS 1.5版本之前只支持Point类型对象。关于椭球面参考系参考函数 ST\_Length\_Spheroid. PostGIS 1.5版本之前只支持Point类型.

**注意**

该函数目前不看几何对象的SRID值，并假定输入的是椭球面坐标。该函数早前版本只支持Point类型.

可用版本: 1.5 -支持除了点类型以外的几何对象，早前的版本只支持点类型对象.

**样例**

SELECT round(CAST(

ST\_Distance\_Spheroid(ST\_Centroid(the\_geom), ST\_GeomFromText(’POINT(-118 38)’,4326), ’ ←

SPHEROID["WGS 84",6378137,298.257223563]’)

As numeric),2) As dist\_meters\_spheroid,

round(CAST(ST\_Distance\_Sphere(ST\_Centroid(the\_geom), ST\_GeomFromText(’POINT(-118 38) ←

’,4326)) As numeric),2) As dist\_meters\_sphere,

round(CAST(ST\_Distance(ST\_Transform(ST\_Centroid(the\_geom),32611),

ST\_Transform(ST\_GeomFromText(’POINT(-118 38)’, 4326),32611)) As numeric),2) As

dist\_utm11\_meters

FROM

←

(SELECT ST\_GeomFromText(’LINESTRING(-118.584 38.374,-118.583 38.5)’, 4326) As the\_geom)

as foo;

dist\_meters\_spheroid | dist\_meters\_sphere | dist\_utm11\_meters

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

70454.92 | 70424.47 | 70438.00

←



PostGIS 2.2.0dev 开发手册

264 / 686

**请参考**

ST\_Distance, ST\_Distance\_Sphere

**8.9.25**

**ST\_DFullyWithin**

ST\_DFullyWithin — 如果输入的所有几何对象之间的距离都在一个指定的距离内，那么返回TRUE

**用法**

boolean ST\_DFullyWithin(geometry g1, geometry g2, double precision distance);

**描述**

如果几何对象的距离完全在另一个指定的距离值范围内，则返回TRUE。指定的距离单位是SRS表中规定的单位。该函数的输入几何对象必须都在同一个坐标投影中，有相同的SRID

**注意**

该函数会自动生成一个bounding box以便充分利用几何对象上的任意索引.

可用版本: 1.5.0

**样例**

postgis=# SELECT ST\_DFullyWithin(geom\_a, geom\_b, 10) as DFullyWithin10, ST\_DWithin(geom\_a,

←

geom\_b, 10) as DWithin10, ST\_DFullyWithin(geom\_a, geom\_b, 20) as DFullyWithin20 from

(select ST\_GeomFromText(’POINT(1 1)’) as geom\_a,ST\_GeomFromText(’LINESTRING(1 5, 2 7, 1 ←

9, 14 12)’) as geom\_b) t1;

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

DFullyWithin10 | DWithin10 | DFullyWithin20 |

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

f

| t

| t

|

**请参考**

ST\_MaxDistance, ST\_DWithin

**8.9.26**

**ST\_DWithin**

ST\_DWithin — 如果输入的几何对象的都在以一个点为中心的指定的距离范围内则返回TRUE。对于geometry类型对象，单位以SRS表中的距离单位为标准。如果是geography类型，单位是米。默认的计算方式是use\_spheroid=true (以椭球面为参考系), 如果想更快的检索出所需要的点，使用use\_spheroid=false，该参数表示以正规的球体为参考系。.

**用法**

boolean ST\_DWithin(geometry g1, geometry g2, double precision distance\_of\_srid);

boolean ST\_DWithin(geography gg1, geography gg2, double precision distance\_meters);

boolean ST\_DWithin(geography gg1, geography gg2, double precision distance\_meters, boolean use\_spheroid);



PostGIS 2.2.0dev 开发手册

265 / 686

**描述**

如果输入的几何对象的都在以一个点为中心的指定的距离范围内则返回TRUE。对于geometry类型对象，单位以SRS表中的距离单位为标准。如果是geography类型，单位是米。该函数只有输入对象是同一个投影系中，有相同的SRID才有意义。如果是geography类型，单位是米。默认的计算方式是use\_spheroid=true (以WGS 84椭球面为参考系), 如果想更快的检索出所需要的点，使用use\_spheroid=false，该参数表示以正规的球体为参考系。.

**注意**

该函数会自动生成一个bounding box以便充分利用几何对象上的任意索引.

**注意**

在1.3版本之前，ST\_Expand是常和&&以及函数ST\_Distance一起使用能达到相同的效果，1.3.4版本及之前，该函数主要用于构建简单的查询。从1.3.4版本开始，相比较之前的版本，ST\_Dwithin函数会使用一个更短的距离函数来更高效地处理更大的缓冲区

**注意**

如果输入参数是3D 几何对象，使用函数ST\_3DDWithin

该函数方法实现了规范 [OpenGIS Simple Features Implementation Speciﬁcation for SQL 1.1.](http://www.opengeospatial.org/standards/sfs)

可用版本: 1.5.0引入支持geography对象

版本提升: 2.1.0 提高处理geography对象的速度. 参考资料 Making Geography faster.

版本提升: 2.1.0 引入对curve几何类型对象的支持.

**样例**

--Find the nearest hospital to each school

--that is within 3000 units of the school.

-- We do an ST\_DWithin search to utilize indexes to limit our search list

-- that the non-indexable ST\_Distance needs to process

--If the units of the spatial reference is meters then units would be meters

SELECT DISTINCT ON (s.gid) s.gid, s.school\_name, s.the\_geom, h.hospital\_name

FROM schools s

LEFT JOIN hospitals h ON ST\_DWithin(s.the\_geom, h.the\_geom, 3000)

ORDER BY s.gid, ST\_Distance(s.the\_geom, h.the\_geom);

--The schools with no close hospitals

--Find all schools with no hospital within 3000 units

--away from the school.

Units is in units of spatial ref (e.g. meters, feet, degrees)

SELECT s.gid, s.school\_name

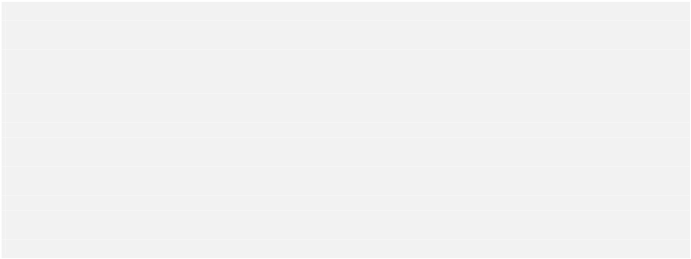
FROM schools s

LEFT JOIN hospitals h ON ST\_DWithin(s.the\_geom, h.the\_geom, 3000)

WHERE h.gid IS NULL;

**请参考**

ST\_Distance, ST\_Expand



PostGIS 2.2.0dev 开发手册

266 / 686

**8.9.27**

**ST\_Equals**

ST\_Equals — 返回给定的几何对象是否是相同的几何对象，不考虑几何对象之间的方向差异（译者注：比如有一个几何对象，将该几何对象绕着其中一个顶点旋转20度，得到一个新的几何对象，那么这个旋转后的几何对象和原来的几何对象是相同的，因为不考虑方向）

**用法**

boolean ST\_Equals(geometry A, geometry B);

**描述**

返回给定的几何对象是否是相同的几何对象，不考虑几何对象之间的方向差异。该函数的返回结果比等号’=’运算符的结果”更好”（译者注：这里的更好应该更符合常规的理解，就像Java里面比较字符串一样）。注意空间意义上的相同意味着下面等式的成立： ST\_Within(A,B) = true 和 ST\_Within(B,A) = true ，同时也意味着几何对象的点的排序可能是不同的，但是表示相同的几何结构。为了确保点的顺序是一致的，使用函数ST\_OrderingEquals (必须注意的是ST\_OrderingEquals比简单验证点的顺序是否相同要更加严格).

**重要**

如果输入的任意一个几何对象是无效的，那么即便这两个几何对象是节上相同，也会返回FALSE

该函数方法实现了规范 OpenGIS Simple Features Implementation Speciﬁcation for SQL 1.1. s2.1.1.2

该函数方法实现了规范 SQL/MM speciﬁcation. SQL-MM 3: 5.1.24

**样例**

SELECT ST\_Equals(ST\_GeomFromText(’LINESTRING(0 0, 10 10)’),

ST\_GeomFromText(’LINESTRING(0 0, 5 5, 10 10)’));

st\_equals

-----------

t

(1 row)

SELECT ST\_Equals(ST\_Reverse(ST\_GeomFromText(’LINESTRING(0 0, 10 10)’)),

ST\_GeomFromText(’LINESTRING(0 0, 5 5, 10 10)’));

st\_equals

-----------

t

(1 row)

**请参考**

ST\_IsValid, ST\_OrderingEquals, ST\_Reverse, ST\_Within

**8.9.28**

**ST\_HasArc**

ST\_HasArc —如果一个geometry对象或geometry collection包含一个CircularString对象，则返回TRUE

**用法**

boolean ST\_HasArc(geometry geomA);



PostGIS 2.2.0dev 开发手册

267 / 686

**描述**

如果一个geometry对象或geometry collection包含一个CircularString对象，则返回TRUE

可用版本: 1.2.3+

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

该函数支持CircularString和Curve几何类型对象

**样例**

SELECT ST\_HasArc(ST\_Collect(’LINESTRING(1 2, 3 4, 5 6)’, ’CIRCULARSTRING(1 1, 2 3, 4 5, 6

7, 5 6)’));

st\_hasarc

--------

t

**请参考**

ST\_CurveToLine, ST\_LineToCurve

←

**8.9.29**

**ST\_Intersects**

ST\_Intersects —如果Geometries/Geography在2维空间内有相交（有共同的空间部分），则返回TRUE。如果他们不相交，那么返回FALSE。对于geography类型对象—误差是0.00001米（因此许多很靠近的点会被认为是相交（即重合））

**用法**

boolean ST\_Intersects( geometry geomA , geometry geomB );

boolean ST\_Intersects( geography geogA , geography geogB );

**描述**

如果Geometries/Geography在2维空间内有相交（有共同的空间部分），则返回TRUE。如果他们不相交，那么返回FALSE。对于geography类型对象—误差是0.00001米（因此许多很靠近的点会被认为是相交（即重合））。

覆盖、接触（相切）、包含都属于空间相交。上述的任意一种都返回TRUE，那么这些几何对象也是空间相交的。空间不相交则返回FALSE

**重要**

对于输入类型是geometry类型的形式（译者注：即上述用法中的第一种形式），该函数不支持GEOMETRYCOLLECTION类型对象作为输入参数. 输入参数是geography类型的形式支持GEOMETRYCOLLECTION，因为这是一个基于距离计算实现的简单包装函数

该函数对于geometry类型，是基于GEOS 模块实现的，对于geography类型，是原生的

可用版本: 1.5引入支持geography对象.



PostGIS 2.2.0dev 开发手册

268 / 686

**注意**

该函数会自动生成一个bounding box以便充分利用几何对象上的任意索引.

**注意**

对于geography对象，该函数的距离误差是0.00001米，并且使用球面参考系而不是椭球参考系来计算

**注意**

这是一个返回逻辑值而不是整型值的可用版本.

该函数方法实现了规范 OpenGIS Simple Features Implementation Speciﬁcation for SQL 1.1. s2.1.1.2 //s2.1.13.3 -

ST\_Intersects(g1, g2 ) --> Not (ST\_Disjoint(g1, g2 )) 两者等价

该函数方法实现了规范 SQL/MM speciﬁcation. SQL-MM 3: 5.1.27

该方法由SFCGAL库提供后台支持.

**Geometry 样例**

SELECT ST\_Intersects(’POINT(0 0)’::geometry, ’LINESTRING ( 2 0, 0 2 )’::geometry);

st\_intersects

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

f

(1 row)

SELECT ST\_Intersects(’POINT(0 0)’::geometry, ’LINESTRING ( 0 0, 0 2 )’::geometry);

st\_intersects

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

t

(1 row)

**Geography 样例**

SELECT ST\_Intersects(

ST\_GeographyFromText(’SRID=4326;LINESTRING(-43.23456 72.4567,-43.23456 72.4568)’),

ST\_GeographyFromText(’SRID=4326;POINT(-43.23456 72.4567772)’)

);

st\_intersects

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

t

**请参考**

ST\_3DIntersects, ST\_Disjoint



PostGIS 2.2.0dev 开发手册

269 / 686

**8.9.30**

**ST\_Length**

ST\_Length — 返回一个LINESTRING或MULTILINESTRING类型对象的2维长度。长度单位对于geometry类型对象来说以空间参考系规定的单位为标准，geography类型以椭圆参考系为参考系，单位是米

**用法**

ﬂoat ST\_Length(geometry a\_2dlinestring);

ﬂoat ST\_Length(geography geog, boolean use\_spheroid=true);

**描述**

对于geometry类型对象来说，返回一个LINESTRING或MULTILINESTRING、ST\_Curve, ST\_MultiCurve返回值对应的类型对象的2维笛卡尔长度（译者注：所谓笛卡尔长度其实就是欧几里得算法的长度）。对于区域面类型的几何对象，该函数返回0，要想计算区域面类型的长度，需要使用函数ST\_Perimeter。长度单位对于geometry类型对象来说以空间参考系规定的单位为标准。对于geography类型对象，单位是米，使用ST\_Perimeter函数来计算区域面类型的对象长度。当前对于几何类型对象来说，该函数是ST\_Length2D别名，但是将来会对更高维度几何对象进行支持。

**警告**

版本变更: 2.0.0 该版本重大突破 –先前版本该函数对于MULTI/POLYGON类型的geography对象，该函数会返回POLYGON/MULTIPOLYGON类型对象的周长。在2.0.0版本，该函数会返回0，以便于geometry类型对象保持一致。如果你想计算一个polygon类型对象的周长，使用函数 ST\_Perimeter

**注意**

对于geography类型对象，该函数默认使用椭球参考系来计算，如果想计算更快但精度差异性的，使用函数ST\_Length(gg,false);

该函数方法实现了规范 OpenGIS Simple Features Implementation Speciﬁcation for SQL 1.1. s2.1.5.1

该函数方法实现了规范 SQL/MM speciﬁcation. SQL-MM 3: 7.1.2, 9.3.4

可用版本: 1.5.0 引入支持geography对象 in 1.5.

该方法由SFCGAL库提供后台支持.

**Geometry 样例**

Return length in feet for line string. NOTE this is in feet because 2249 is Mass State Plane Feet

SELECT ST\_Length(ST\_GeomFromText(’LINESTRING(743238 2967416,743238 2967450,743265 2967450,

743265.625 2967416,743238 2967416)’,2249));

st\_length

---------

122.630744000095

--Transforming WGS 84 linestring to Massachusetts state plane meters

SELECT ST\_Length(

ST\_Transform(

ST\_GeomFromEWKT(’SRID=4326;LINESTRING(-72.1260 42.45, -72.1240 42.45666, -72.123

42.1546)’),

←



PostGIS 2.2.0dev 开发手册

270 / 686

26986

)

);

st\_length

---------

34309.4563576191

**Geography 样例**

Return length of WGS 84 geography line

-- default calculation is using a sphere rather than spheroid

SELECT ST\_Length(the\_geog) As length\_spheroid,

ST\_Length(the\_geog,false) As length\_sphere

FROM (SELECT ST\_GeographyFromText(

’SRID=4326;LINESTRING(-72.1260 42.45, -72.1240 42.45666, -72.123 42.1546)’) As the\_geog)

As foo;

length\_spheroid

|

length\_sphere

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

34310.5703627305 | 34346.2060960742

(1 row)

**请参考**

ST\_GeographyFromText, ST\_GeomFromEWKT, ST\_Length\_Spheroid, ST\_Perimeter, ST\_Transform

**8.9.31**

**ST\_Length2D**

ST\_Length2D —返回一个LINESTRING或MULTILINESTRING类型对象的2维长度，该函数是函数ST\_Length的别名

**用法**

ﬂoat ST\_Length2D(geometry a\_2dlinestring);

**描述**

返回一个LINESTRING或MULTILINESTRING类型对象的2维长度，该函数是函数ST\_Length的别名

**请参考**

ST\_Length, ST\_3DLength

**8.9.32**

**ST\_3DLength**

ST\_3DLength —返回一个LINESTRING或MULTILINESTRING类型对象的3维或2维长度。

**用法**

ﬂoat ST\_3DLength(geometry a\_3dlinestring);

PostGIS 2.2.0dev 开发手册

271 / 686

**描述**

返回一个LINESTRING或MULTILINESTRING类型对象的3维或2维长度。对于2维的LINESTRING类型对象，该函数只会返回其2维长度 (与函数ST\_Length 和 ST\_Length2D返回值一样)

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

版本变更: 2.0.0 在早先的版本中，该函数名字是ST\_Length3D（尼玛，城会玩。。。）

**样例**

Return length in feet for a 3D cable. NOTE this is in feet because 2249 is Mass State Plane Feet

SELECT ST\_3DLength(ST\_GeomFromText(’LINESTRING(743238 2967416 1,743238 2967450 1,743265

2967450 3,

743265.625 2967416 3,743238 2967416 3)’,2249));

ST\_3DLength

-----------

122.704716741457

**请参考**

ST\_Length, ST\_Length2D

←

**8.9.33**

**ST\_Length\_Spheroid**

ST\_Length\_Spheroid —在椭圆参考系中计算一个linestring/multilinestring的2D或3D长度，如果几何类型对象的坐标是经纬度形式，并且长度不需要重投影的话，使用该函数很有用

**用法**

ﬂoat ST\_Length\_Spheroid(geometry a\_linestring, spheroid a\_spheroid);

**描述**

在椭圆参考系中计算一个linestring/multilinestring的2D或3D长度，如果几何类型对象的坐标是经纬度形式，并且长度不需要重投影的话，使用该函数很有用。椭圆面是一个单独的数据库类型对象，构建方法如下：

SPHEROID[<NAME>,<SEMI-MAJOR

AXIS>,<INVERSE FLATTENING>]

SPHEROID["GRS\_1980",6378137,298.257222101]

**注意**

如果输入的几何类型对象不是MULTILINESTRING or LINESTRING返回0

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.



PostGIS 2.2.0dev 开发手册

272 / 686

**样例**

SELECT ST\_Length\_Spheroid( geometry\_column,

’SPHEROID["GRS\_1980",6378137,298.257222101]’ )

FROM geometry\_table;

SELECT ST\_Length\_Spheroid( the\_geom, sph\_m ) As tot\_len,

ST\_Length\_Spheroid(ST\_GeometryN(the\_geom,1), sph\_m) As len\_line1,

ST\_Length\_Spheroid(ST\_GeometryN(the\_geom,2), sph\_m) As len\_line2

FROM (SELECT ST\_GeomFromText(’MULTILINESTRING((-118.584 38.374,-118.583 38.5),

(-71.05957 42.3589 , -71.061 43))’) As the\_geom,

CAST(’SPHEROID["GRS\_1980",6378137,298.257222101]’ As spheroid) As sph\_m)

as foo;

tot\_len

|

len\_line1

|

len\_line2

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

85204.5207562955 | 13986.8725229309 | 71217.6482333646

--3D

SELECT ST\_Length\_Spheroid( the\_geom, sph\_m ) As tot\_len,

ST\_Length\_Spheroid(ST\_GeometryN(the\_geom,1), sph\_m) As len\_line1,

ST\_Length\_Spheroid(ST\_GeometryN(the\_geom,2), sph\_m) As len\_line2

FROM (SELECT ST\_GeomFromEWKT(’MULTILINESTRING((-118.584 38.374 20,-118.583 38.5 30) ←

,

(-71.05957 42.3589 75, -71.061 43 90))’) As the\_geom,

CAST(’SPHEROID["GRS\_1980",6378137,298.257222101]’ As spheroid) As sph\_m)

as foo;

tot\_len

|

len\_line1

|

len\_line2

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

85204.5259107402 | 13986.876097711 | 71217.6498130292

**请参考**

ST\_GeometryN, ST\_Length, ST\_3DLength\_Spheroid

**8.9.34**

**ST\_Length2D\_Spheroid**

ST\_Length2D\_Spheroid —在椭圆参考系中计算一个linestring/multilinestring的2D长度，如果几何类型对象的坐标是经纬度形式，并且长度不需要重投影的话，使用该函数很有用

**用法**

ﬂoat ST\_Length2D\_Spheroid(geometry a\_linestring, spheroid a\_spheroid);

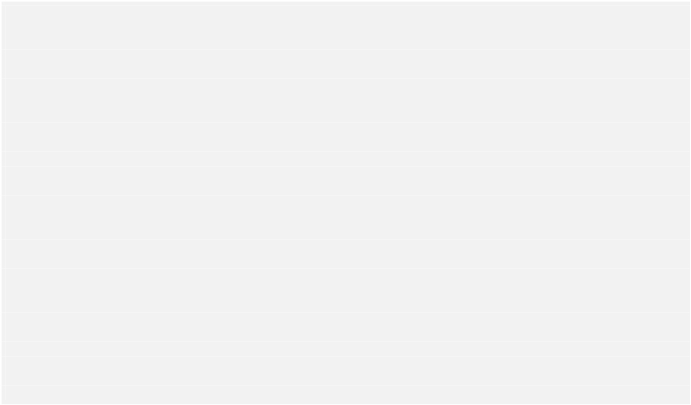
**描述**

在椭圆参考系中计算一个linestring/multilinestring的2D长度，如果几何类型对象的坐标是经纬度形式，并且长度不需要重投影的话，使用该函数很有用。椭圆面是一个单独的数据库类型对象，构建方法如下：

SPHEROID[<NAME>,<SEMI-MAJOR

AXIS>,<INVERSE FLATTENING>]

SPHEROID["GRS\_1980",6378137,298.257222101]



PostGIS 2.2.0dev 开发手册

273 / 686

**注意**

如果输入参数不是MULTILINESTRING或LINESTRING，返回值都是0

**注意**

该函数与函数ST\_Length\_Spheroid 和函数 ST\_3DLength\_Spheroid 都很类似，区别在于ST\_Length2D\_Spheroid在计算中会丢弃Z坐标

**样例**

SELECT ST\_Length2D\_Spheroid( geometry\_column,

’SPHEROID["GRS\_1980",6378137,298.257222101]’ )

FROM geometry\_table;

SELECT ST\_Length2D\_Spheroid( the\_geom, sph\_m ) As tot\_len,

ST\_Length2D\_Spheroid(ST\_GeometryN(the\_geom,1), sph\_m) As len\_line1,

ST\_Length2D\_Spheroid(ST\_GeometryN(the\_geom,2), sph\_m) As len\_line2

FROM (SELECT ST\_GeomFromText(’MULTILINESTRING((-118.584 38.374,-118.583 38.5),

(-71.05957 42.3589 , -71.061 43))’) As the\_geom,

CAST(’SPHEROID["GRS\_1980",6378137,298.257222101]’ As spheroid) As sph\_m)

as foo;

tot\_len

|

len\_line1

|

len\_line2

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

85204.5207562955 | 13986.8725229309 | 71217.6482333646

--3D Observe same answer

SELECT ST\_Length2D\_Spheroid( the\_geom, sph\_m ) As tot\_len,

ST\_Length2D\_Spheroid(ST\_GeometryN(the\_geom,1), sph\_m) As len\_line1,

ST\_Length2D\_Spheroid(ST\_GeometryN(the\_geom,2), sph\_m) As len\_line2

FROM (SELECT ST\_GeomFromEWKT(’MULTILINESTRING((-118.584 38.374 20,-118.583 38.5 30) ←

,

(-71.05957 42.3589 75, -71.061 43 90))’) As the\_geom,

CAST(’SPHEROID["GRS\_1980",6378137,298.257222101]’ As spheroid) As sph\_m)

as foo;

tot\_len

|

len\_line1

|

len\_line2

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

85204.5207562955 | 13986.8725229309 | 71217.6482333646

**请参考**

ST\_GeometryN, ST\_Length\_Spheroid, ST\_3DLength\_Spheroid

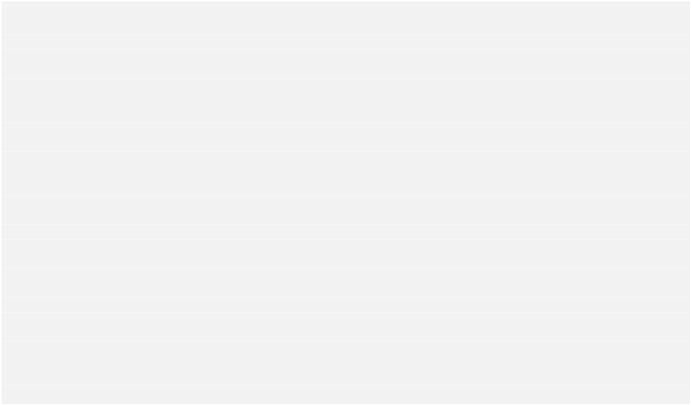
**8.9.35**

**ST\_3DLength\_Spheroid**

ST\_3DLength\_Spheroid — 返回一个椭圆面的几何类型对象的长度，该函数会考虑到高度坐标，该函数只是函数ST\_Length\_Spheroid的别名。

**用法**

ﬂoat ST\_3DLength\_Spheroid(geometry a\_linestring, spheroid a\_spheroid);



PostGIS 2.2.0dev 开发手册

274 / 686

**描述**

返回一个椭圆面的几何类型对象的长度，该函数会考虑到高度坐标，该函数只是函数ST\_Length\_Spheroid的别名。

**注意**

版本变更: 2.0.0 。在之前的版本中，对于任意非MULTILINESTRING 金额 LINESTRING类型对象，该函数会返回0.在2.0.0版本中该函数对于任意的polygon类型对象，会返回它的周长。

**注意**

该函数只是函数ST\_Length\_Spheroid的别名。

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

版本变更: 2.0.0 在之前的版本中，该函数名为ST\_Length3d\_Spheroid

**样例**

See ST\_Length\_Spheroid

**请参考**

ST\_GeometryN, ST\_Length, ST\_Length\_Spheroid

**8.9.36**

**ST\_LongestLine**

ST\_LongestLine — 返回两个几何对象之间的2维最长距离。如果有多个线段都是最长线段，那么返回第一个最长的。该线段的起点在g1，终点在g2.该函数返回的线段的长度与用函数ST\_MaxDistance计算g1和g2的距离值一样。

**用法**

geometry ST\_LongestLine(geometry g1, geometry g2);

**描述**

返回两个几何对象之间的2维最长距离。

可用版本: 1.5.0

**样例**



PostGIS 2.2.0dev 开发手册

275 / 686

longest line between polygon and polygon

Longest line between point and line

SELECT ST\_AsText(

ST\_LongestLine(’POINT(100 100)’:: ←

geometry,

’LINESTRING (20 80, 98 ←

190, 110 180, 50 75 )’::geometry)

) As lline;

SELECT ST\_AsText(

ST\_LongestLine(

ST\_GeomFromText(’POLYGON ←

((175 150, 20 40,

50 60, 125 100, ←

175 150))’),

ST\_Buffer(ST\_GeomFromText ←

(’POINT(110 170)’), 20)

)

) As llinewkt;

lline

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

LINESTRING(100 100,98 190)

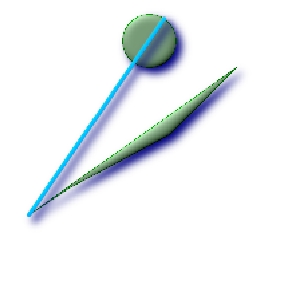
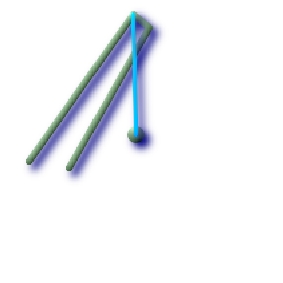
lline

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

LINESTRING(20 40,121.111404660392

186.629392246051)

←



PostGIS 2.2.0dev 开发手册

276 / 686

longest straight distance to travel from one part of an elegant city to the other NOTE the max distance = to the length of the

line.

SELECT ST\_AsText(ST\_LongestLine(c.the\_geom, c.the\_geom)) As llinewkt,

ST\_MaxDistance(c.the\_geom,c.the\_geom) As max\_dist,

ST\_Length(ST\_LongestLine(c.the\_geom, c.the\_geom)) As lenll

FROM (SELECT ST\_BuildArea(ST\_Collect(the\_geom)) As the\_geom

FROM (SELECT ST\_Translate(ST\_SnapToGrid(ST\_Buffer(ST\_Point(50 ,generate\_series ←

(50,190, 50)

),40, ’quad\_segs=2’),1), x, 0)

As the\_geom

) As c;

FROM generate\_series(1,100,50) As x)

AS foo

llinewkt

|

max\_dist

|

lenll

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

LINESTRING(23 22,129 178) | 188.605408193933 | 188.605408193933

**请参考**

ST\_MaxDistance, ST\_ShortestLine, ST\_LongestLine

**8.9.37**

**ST\_OrderingEquals**

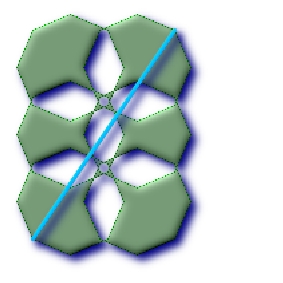
ST\_OrderingEquals —如果两个输入的几何对象表示相同的几何对象，并且几何对象的点的顺序也是一致的，那么返回值为TRUE

**用法**

boolean ST\_OrderingEquals(geometry A, geometry B);

**描述**

如果两个输入的几何对象表示相同的几何对象，并且几何对象的点的顺序也是一致的，那么返回值为TRUE，否则返回FALSE



PostGIS 2.2.0dev 开发手册

277 / 686

**注意**

该函数是按照ArcSDE的SQL 规范而不是SQL-MM的规范实现的，参考：

http://edndoc.esri.com/arcsde/9.1/sql\_api/sqlapi3.htm#ST\_OrderingEquals

该函数方法新版本也实现了规范 SQL/MM speciﬁcation. SQL-MM 3: 5.1.43

**样例**

SELECT ST\_OrderingEquals(ST\_GeomFromText(’LINESTRING(0 0, 10 10)’),

ST\_GeomFromText(’LINESTRING(0 0, 5 5, 10 10)’));

st\_orderingequals

-----------

f

(1 row)

SELECT ST\_OrderingEquals(ST\_GeomFromText(’LINESTRING(0 0, 10 10)’),

ST\_GeomFromText(’LINESTRING(0 0, 0 0, 10 10)’));

st\_orderingequals

-----------

t

(1 row)

SELECT ST\_OrderingEquals(ST\_Reverse(ST\_GeomFromText(’LINESTRING(0 0, 10 10)’)),

ST\_GeomFromText(’LINESTRING(0 0, 0 0, 10 10)’));

st\_orderingequals

-----------

f

(1 row)

注意：上面的样例是手册原文给出的，可能和实际的版本结果不一定一致。译者给出9.5版本的PostgreSQL和

2.2版本的PostGIS环境下的两个样例：

**SELECT** ST\_OrderingEquals(ST\_GeomFromText('LINESTRING(0 0, 10 10)'),

ST\_GeomFromText('LINESTRING(10 10, 0 0)'));

返回false

**SELECT** ST\_OrderingEquals(ST\_GeomFromText('LINESTRING(0 0, 10 10)'),

ST\_GeomFromText('LINESTRING(0 0, 10 10)'));

返回true

**SELECT** ST\_OrderingEquals(ST\_GeomFromText('LINESTRING(0 0, 10 10)'),

ST\_GeomFromText('LINESTRING(0 0, 0 0, 10 10)'));

返回false

**请参考**

ST\_Equals, ST\_Reverse

**8.9.38**

**ST\_Overlaps**

ST\_Overlaps — 如果几何对象之间有共同的空间，相同的维度，但是互相之间没有相互包含，那么返回TRUE

**用法**

boolean ST\_Overlaps(geometry A, geometry B);

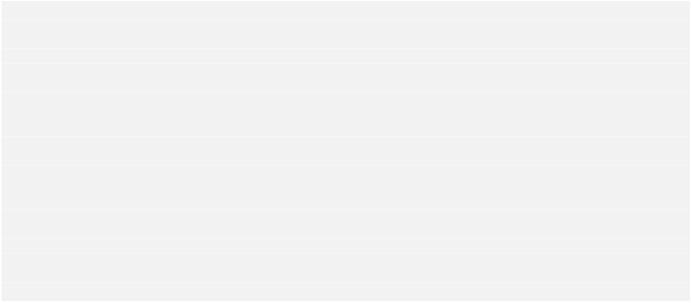
**描述**

如果两个几何对象有空间覆盖，就是两个几何对象之间有交集，但互相之间没有相互包含。

该函数由GEOS模块支持

**注意**

该函数不支持GeometryCollection参数



PostGIS 2.2.0dev 开发手册

278 / 686

该函数使用时候会自动生成一个bounding box，以便充分利用几何对象上的任意一个索引. 如果想避免使用索引，使用函数\_ST\_Overlaps.这是一个返回逻辑值而不是整型值的可用版本.

该函数方法实现了规范 OpenGIS Simple Features Implementation Speciﬁcation for SQL 1.1. s2.1.1.2 // s2.1.13.3

该函数方法实现了规范 SQL/MM speciﬁcation. SQL-MM 3: 5.1.32

**样例**

下面的情况都返回TRUE

MULTIPOINT / MULTIPOINT

LINESTRING / LINESTRING

POLYGON / POLYGON

--a point on a line is contained by the line and is of a lower dimension, and therefore

does not overlap the line

nor crosses

SELECT ST\_Overlaps(a,b) As a\_overlap\_b,

ST\_Crosses(a,b) As a\_crosses\_b,

ST\_Intersects(a, b) As a\_intersects\_b, ST\_Contains(b,a) As b\_contains\_a

←

FROM (SELECT ST\_GeomFromText(’POINT(1 0.5)’) As a, ST\_GeomFromText(’LINESTRING(1 0, 1 1, 3

←

5)’)

As b)

As foo

a\_overlap\_b | a\_crosses\_b | a\_intersects\_b | b\_contains\_a

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

f

| f

| t

| t

--a line that is partly contained by circle, but not fully is defined as intersecting and

crossing,

-- but since of different dimension it does not overlap

SELECT ST\_Overlaps(a,b) As a\_overlap\_b, ST\_Crosses(a,b) As a\_crosses\_b,

ST\_Intersects(a, b) As a\_intersects\_b,

ST\_Contains(a,b) As a\_contains\_b

←

FROM (SELECT ST\_Buffer(ST\_GeomFromText(’POINT(1 0.5)’), 3)

As a, ST\_GeomFromText(’ ←

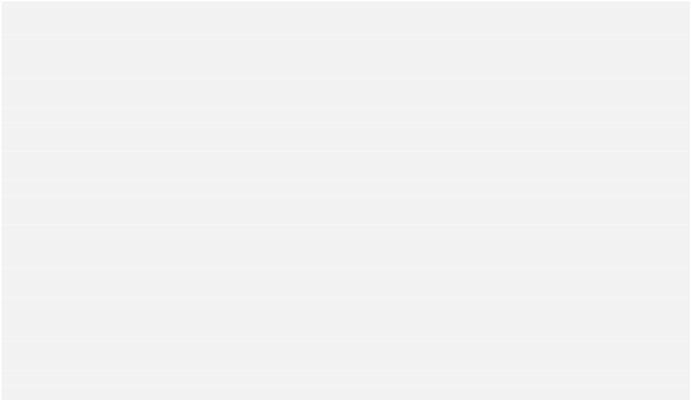
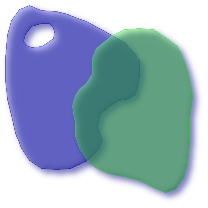
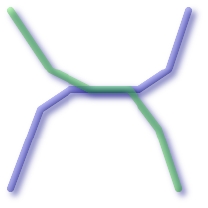
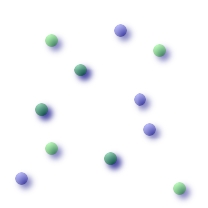
LINESTRING(1 0, 1 1, 3 5)’)

As b)

As foo;

a\_overlap\_b | a\_crosses\_b | a\_intersects\_b | a\_contains\_b

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



PostGIS 2.2.0dev 开发手册

279 / 686

f

| t

| t

| f

-- a 2-dimensional bent hot dog (aka buffered line string) that intersects a circle,

-- but is not fully contained by the circle is defined as overlapping since they are of

the same dimension,

←

--

--

but it does not cross, because the intersection of the 2 is of the same dimension

as the maximum dimension of the 2

SELECT ST\_Overlaps(a,b) As a\_overlap\_b, ST\_Crosses(a,b) As a\_crosses\_b, ST\_Intersects(a, b) ←

As a\_intersects\_b,

ST\_Contains(b,a) As b\_contains\_a,

ST\_Dimension(a) As dim\_a, ST\_Dimension(b) as dim\_b, ST\_Dimension(ST\_Intersection(a,b)) As

dima\_intersection\_b

←

FROM (SELECT ST\_Buffer(ST\_GeomFromText(’POINT(1 0.5)’), 3)

As a,

ST\_Buffer(ST\_GeomFromText(’LINESTRING(1 0, 1 1, 3 5)’),0.5)

As foo;

As b)

a\_overlap\_b | a\_crosses\_b | a\_intersects\_b | b\_contains\_a | dim\_a | dim\_b |

←

dima\_intersection\_b

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

t

| f

| t

| f

|

2 |

2 |

2

**请参考**

ST\_Contains, ST\_Crosses, ST\_Dimension, ST\_Intersects

**8.9.39**

**ST\_Perimeter**

ST\_Perimeter — 返回ST\_Surface 或 ST\_MultiSurface (返回Polygon, Multipolygon类型对象) 值对应的geometry 和 geography类型对象的周长。对于geometry类型对象，返回的长度以空间参考系规定的单位做标准，对于geography类型对象，单位是米

**用法**

ﬂoat ST\_Perimeter(geometry g1);

ﬂoat ST\_Perimeter(geography geog, boolean use\_spheroid=true);

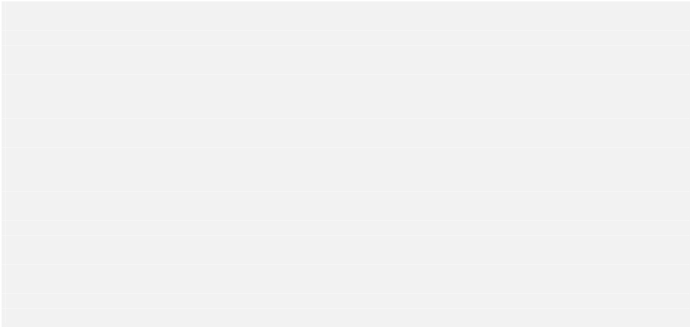
**描述**

返回ST\_Surface 或 ST\_MultiSurface (返回Polygon, Multipolygon类型对象) 值对应的geometry 和 geography类型对象的2D 周长。对于非表面型几何类型对象，返回0。对于LINESTRING类型对象，使用函数ST\_Length。对于geometry类型对象，返回的长度以空间参考系规定的单位做标准，对于geography类型对象，单位是米。如果参数use\_spheroid值为false，那么该函数会使用地球作为球面参考系，而不是椭圆参考系。当前该函数是函数ST\_Perimeter2D的别名，但将来该函数会支持更高维度的对象

该函数方法实现了规范 OpenGIS Simple Features Implementation Speciﬁcation for SQL 1.1. s2.1.5.1

该函数方法实现了规范 SQL/MM speciﬁcation. SQL-MM 3: 8.1.3, 9.5.4

可用性 2.0.0:引入支持geography对象



PostGIS 2.2.0dev 开发手册

280 / 686

**样例: Geometry**

Return perimeter in feet for polygon and multipolygon. 注意 this is in feet because 2249 is Mass State Plane Feet

SELECT ST\_Perimeter(ST\_GeomFromText(’POLYGON((743238 2967416,743238 2967450,743265 2967450,

743265.625 2967416,743238 2967416))’, 2249));

st\_perimeter

---------

122.630744000095

(1 row)

SELECT ST\_Perimeter(ST\_GeomFromText(’MULTIPOLYGON(((763104.471273676 2949418.44119003,

763104.477769673 2949418.42538203,

763104.189609677 2949418.22343004,763104.471273676 2949418.44119003)),

((763104.471273676 2949418.44119003,763095.804579742 2949436.33850239,

763086.132105649 2949451.46730207,763078.452329651 2949462.11549407,

763075.354136904 2949466.17407812,763064.362142565 2949477.64291974,

763059.953961626 2949481.28983009,762994.637609571 2949532.04103014,

762990.568508415 2949535.06640477,762986.710889563 2949539.61421415,

763117.237897679 2949709.50493431,763235.236617789 2949617.95619822,

763287.718121842 2949562.20592617,763111.553321674 2949423.91664605,

763104.471273676 2949418.44119003)))’, 2249));

st\_perimeter

---------

845.227713366825

(1 row)

**样例: Geography**

Return perimeter in meters and feet for polygon and multipolygon. NOTE this is geography (WGS 84 long lat)

SELECT

ST\_Perimeter(geog) As per\_meters, ST\_Perimeter(geog)/0.3048 As per\_ft

FROM ST\_GeogFromText(’POLYGON((-71.1776848522251 42.3902896512902,-71.1776843766326

42.3903829478009,

←

-71.1775844305465 42.3903826677917,-71.1775825927231 42.3902893647987,-71.1776848522251

42.3902896512902))’) As geog;

←

per\_meters

|

per\_ft

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

37.3790462565251 | 122.634666195949

-- Multipolygon example --

SELECT

ST\_Perimeter(geog) As per\_meters, ST\_Perimeter(geog,false) As per\_sphere\_meters,

←

ST\_Perimeter(geog)/0.3048 As per\_ft

FROM ST\_GeogFromText(’MULTIPOLYGON(((-71.1044543107478 42.340674480411,-71.1044542869917

42.3406744369506,

-71.1044553562977 42.340673886454,-71.1044543107478 42.340674480411)),

((-71.1044543107478 42.340674480411,-71.1044860600303 42.3407237015564,-71.1045215770124

42.3407653385914,

-71.1045498002983 42.3407946553165,-71.1045611902745 42.3408058316308,-71.1046016507427

42.340837442371,

-71.104617893173 42.3408475056957,-71.1048586153981 42.3409875993595,-71.1048736143677

42.3409959528211,

-71.1048878050242 42.3410084812078,-71.1044020965803 42.3414730072048,

-71.1039672113619 42.3412202916693,-71.1037740497748 42.3410666421308,

-71.1044280218456 42.3406894151355,-71.1044543107478 42.340674480411)))’) As geog;

←

←

←

←

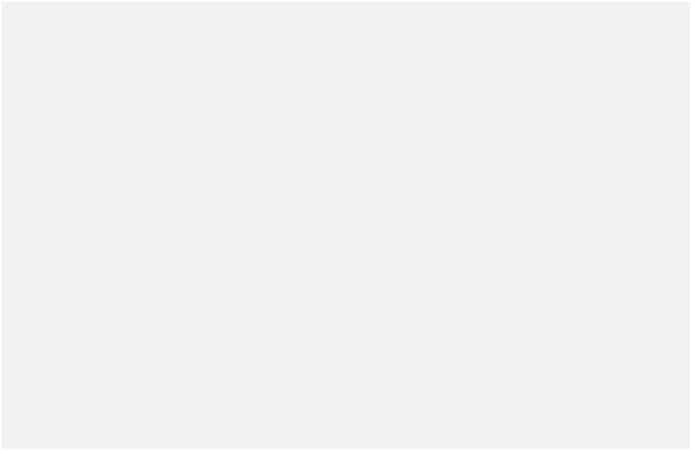
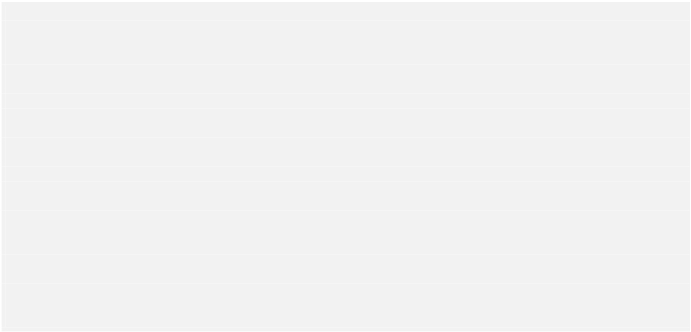
per\_meters

| per\_sphere\_meters |

per\_ft

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

257.634283683311 | 257.412311446337 | 845.256836231335



PostGIS 2.2.0dev 开发手册

281 / 686

**请参考**

ST\_GeogFromText, ST\_GeomFromText, ST\_Length

**8.9.40**

**ST\_Perimeter2D**

ST\_Perimeter2D — 如果输入对象是polygon 或multipolygon对象，则返回2维周长。该函数是函数ST\_Perimeter的别名。

**用法**

ﬂoat ST\_Perimeter2D(geometry geomA);

**描述**

如果输入对象是polygon 或multipolygon对象，则返回2维周长。该函数是函数ST\_Perimeter的别名。

**注意**

该函数是函数ST\_Perimeter的别名。未来版本的ST\_Perimeter会返回一个几何对象的最高维度中的周长。目前该函数的实现还在研究中

**请参考**

ST\_Perimeter

**8.9.41**

**ST\_3DPerimeter**

ST\_3DPerimeter —如果输入对象是polygon 或multipolygon对象，则返回3维周长。

**用法**

ﬂoat ST\_3DPerimeter(geometry geomA);

**描述**

如果输入对象是polygon 或multipolygon对象，则返回3维周长。如果输入对象是2维的几何对象，那么返回的是2维周长

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

版本变更: 2.0.0 In prior versions this used to be called ST\_Perimeter3D



PostGIS 2.2.0dev 开发手册

282 / 686

**样例**

Perimeter of a slightly elevated polygon in the air in Massachusetts state plane feet

SELECT ST\_3DPerimeter(the\_geom), ST\_Perimeter2d(the\_geom), ST\_Perimeter(the\_geom) FROM

(SELECT ST\_GeomFromEWKT(’SRID=2249;POLYGON((743238 2967416 2,743238 2967450 1,

743265.625 2967416 1,743238 2967416 2))’) As the\_geom) As foo;

ST\_3DPerimeter

|

st\_perimeter2d

|

st\_perimeter

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

105.465793597674 | 105.432997272188 | 105.432997272188

**请参考**

ST\_GeomFromEWKT, ST\_Perimeter, ST\_Perimeter2D

**8.9.42**

**ST\_PointOnSurface**

ST\_PointOnSurface — 返回一个必然在Surface对象上的点

**用法**

geometry ST\_PointOnSurface(geometry g1);

**描述**

返回一个必然在Surface对象上的点

该函数方法实现了规范 OpenGIS Simple Features Implementation Speciﬁcation for SQL 1.1. s3.2.14.2 // s3.2.18.2

该函数方法实现了规范 SQL/MM speciﬁcation. SQL-MM 3: 8.1.5, 9.5.6.根据该规范，函数 ST\_PointOnSurface支持Surface类型几何对象(POLYGONs, MULTIPOLYGONS, CURVED POLYGONS)。因此PostGIS似乎扩展了该规范所要求的对象。大多数数据库例如Oracle、DB2、ESRI SDE 似乎只支持Surface类型对象。而SQL Server 2008 和 PostGIS 一样支持所有几何类型对象

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

**样例**

SELECT ST\_AsText(ST\_PointOnSurface(’POINT(0 5)’::geometry));

st\_astext

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

POINT(0 5)

(1 row)

SELECT ST\_AsText(ST\_PointOnSurface(’LINESTRING(0 5, 0 10)’::geometry));

st\_astext

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

POINT(0 5)

(1 row)

SELECT ST\_AsText(ST\_PointOnSurface(’POLYGON((0 0, 0 5, 5 5, 5 0, 0 0))’::geometry));



PostGIS 2.2.0dev 开发手册

283 / 686

st\_astext

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

POINT(2.5 2.5)

(1 row)

SELECT ST\_AsEWKT(ST\_PointOnSurface(ST\_GeomFromEWKT(’LINESTRING(0 5 1, 0 0 1, 0 10 2)’)));

st\_asewkt

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

POINT(0 0 1)

(1 row)

**请参考**

ST\_Centroid, ST\_Point\_Inside\_Circle

**8.9.43**

**ST\_Project**

ST\_Project — 返回一个输入类型为距离（单位：米）和方位角（单位：弧度）经投影转换后的Point类型

**用法**

geography ST\_Project(geography g1, ﬂoat distance, ﬂoat azimuth);

**描述**

返回一个输入类型为距离（单位：米）和方位角（单位：弧度）经投影转换后的Point类型

距离、方位角和投影系统是对两个点操作、描述位置关系的全部要素

方位角有时候被称为航向角。它的值的计算起始轴是正北方向，记为0方位角

东方的方位角是90度，南方的方位角是180度，西方的方位角是270度

距离单位是米

可用版本: 2.0.0

**样例: Using degrees - projected point 100,000 meters and bearing 45 degrees**

SELECT ST\_AsText(ST\_Project(’POINT(0 0)’::geography, 100000, radians(45.0)));

st\_astext

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

POINT(0.63523102912532 0.63947233472882)

(1 row)

**样例: Using radians - projected point 100,000 meters and bearing pi/4 (45 degrees)**

SELECT ST\_AsText(ST\_Project(’POINT(0 0)’::geography, 100000, pi()/4));

st\_astext

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

POINT(0.63523102912532 0.63947233472882)

(1 row)

PostGIS 2.2.0dev 开发手册

284 / 686

**请参考**

ST\_Azimuth, ST\_Distance, [PostgreSQL Math Functions](http://www.postgresql.org/docs/current/interactive/functions-math.html)

**8.9.44**

**ST\_Relate**

ST\_Relate —根据输入参数intersectionMatrixPattern的值，检查两个几何对象的内部、边界、外部是否有相交，如果有，则返回true。

**用法**

boolean ST\_Relate(geometry geomA, geometry geomB, text intersectionMatrixPattern);

text ST\_Relate(geometry geomA, geometry geomB);

text ST\_Relate(geometry geomA, geometry geomB, int BoundaryNodeRule);

**描述**

函数形式 1:输入参数geomA, geomB, intersectionMatrix ，检查两个几何对象的内部、边界、外部是否有相交，如果有返回1 (TRUE) ，参考 [DE-9IM matrix pattern.](http://en.wikipedia.org/wiki/DE-9IM)

对于一步检查两个几何对象的相交，交叉等关系时候很有用

该函数不支持GeometryCollection参数

**注意**

这是一个返回逻辑值而不是整型值的可用版本，并在OGC SPEC规范中有定义

**注意**

该函数不会自动使用索引，因为两个几何对象的关系可能是相反的，例如不相交。如果你正在使用一个需要进行相交计算的查询，使用运算符&&，它会使用索引

形式 2: 输入参数 geomA and geomB ，返回值参考章节4.3.6

形式 3: 和形式2亿元，但是允许指定一个边界点规则 (1:OGC/MOD2, 2:Endpoint, 3:MultivalentEndpoint,4:MonovalentEndpoint)

**注意**

该函数不支持GeometryCollection参数

该函数不在OGC 规范中，但是有在s2.1.13.2节中提到

该函数由GEOS模块支持

该函数方法实现了规范 OpenGIS Simple Features Implementation Speciﬁcation for SQL 1.1. s2.1.1.2 // s2.1.13.3

该函数方法实现了规范 SQL/MM speciﬁcation. SQL-MM 3: 5.1.25

版本提升: 2.0.0 –添加支持指定边界点规则 (需要版本GEOS >= 3.0).



PostGIS 2.2.0dev 开发手册

285 / 686

**样例**

--Find all compounds that intersect and not touch a poly (interior intersects)

SELECT l.\* , b.name As poly\_name

FROM polys As b

INNER JOIN compounds As l

ON (p.the\_geom && b.the\_geom

AND ST\_Relate(l.the\_geom, b.the\_geom,’T\*\*\*\*\*\*\*\*’));

SELECT ST\_Relate(ST\_GeometryFromText(’POINT(1 2)’), ST\_Buffer(ST\_GeometryFromText(’POINT(1

2)’),2));

st\_relate

-----------

0FFFFF212

SELECT ST\_Relate(ST\_GeometryFromText(’LINESTRING(1 2, 3 4)’), ST\_GeometryFromText(’ ←

LINESTRING(5 6, 7 8)’));

st\_relate

-----------

FF1FF0102

SELECT ST\_Relate(ST\_GeometryFromText(’POINT(1 2)’), ST\_Buffer(ST\_GeometryFromText(’POINT(1

2)’),2), ’0FFFFF212’);

st\_relate

-----------

t

SELECT ST\_Relate(ST\_GeometryFromText(’POINT(1 2)’), ST\_Buffer(ST\_GeometryFromText(’POINT(1

2)’),2), ’\*FF\*FF212’);

st\_relate

-----------

t

**请参考**

ST\_Crosses, Section 4.3.6, ST\_Disjoint, ST\_Intersects, ST\_Touches

←

←

←

**8.9.45**

**ST\_RelateMatch**

ST\_RelateMatch — 如果参数intersectionMattrixPattern1 满足了 intersectionMatrixPattern2参数对应的空间关系，则返回TRUE

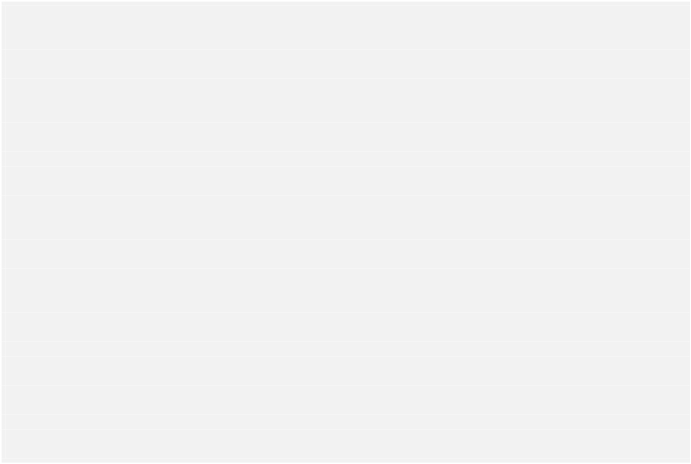
**用法**

boolean ST\_RelateMatch(text intersectionMatrix, text intersectionMatrixPattern);

**描述**

如果参数intersectionMattrixPattern1 满足了 intersectionMatrixPattern2参数对应的空间关系，则返回TRUE。更多信息参考章节4.3.6.

可用版本: 2.0.0 -需要GEOS版本>= 3.3.0.



PostGIS 2.2.0dev 开发手册

286 / 686

**样例**

SELECT ST\_RelateMatch(’101202FFF’, ’TTTTTTFFF’) ;

-- result --

t

--example of common intersection matrix patterns and example matrices

-- comparing relationships of involving one invalid geometry and ( a line and polygon that

intersect at interior and boundary)

SELECT mat.name, pat.name, ST\_RelateMatch(mat.val, pat.val) As satisfied

FROM

( VALUES (’Equality’, ’T1FF1FFF1’),

(’Overlaps’, ’T\*T\*\*\*T\*\*’),

(’Within’, ’T\*F\*\*F\*\*\*’),

(’Disjoint’, ’FF\*FF\*\*\*\*’) As pat(name,val)

CROSS JOIN

( VALUES (’Self intersections (invalid)’, ’111111111’),

(’IE2\_BI1\_BB0\_BE1\_EI1\_EE2’, ’FF2101102’),

(’IB1\_IE1\_BB0\_BE0\_EI2\_EI1\_EE2’, ’F11F00212’)

) As mat(name,val);

**请参考**

Section 4.3.6, ST\_Relate

←

**8.9.46**

**ST\_ShortestLine**

ST\_ShortestLine —返回两个几何对象之间最短的2维线段

**用法**

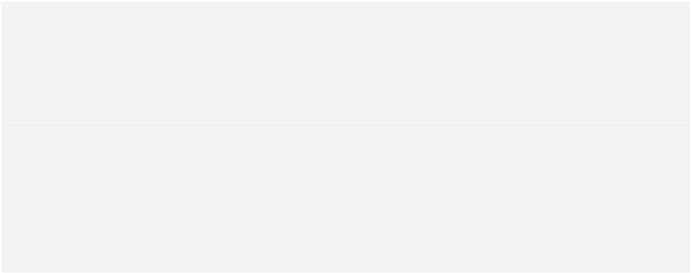
geometry ST\_ShortestLine(geometry g1, geometry g2);

**描述**

返回两个几何对象之间2维最短的线段，如果函数找到这两个几何对象之间不止一条线段，那么只会返回第一条。如果g1和g2只在一个点相交，该函数会返回一条返回起点和终点都在该交点的线段。如果g1和g2相交不止一个点，该函数会返回起点和终点都在同一点的线段，但该交点可以是g1和g2相交的任意一点。返回的线段起点总是属于g1，终点属于g2。该函数返回的线段的3D长度总是和ST\_Distance计算g1和g2之间的距离值是一样的。

可用版本: 1.5.0

**样例**



PostGIS 2.2.0dev 开发手册

287 / 686

Shortest line between point and linestring

shortest line between polygon and polygon

SELECT ST\_AsText(

SELECT ST\_AsText(

ST\_ShortestLine(’POINT(100 100) ←

ST\_ShortestLine(

’::geometry,

’LINESTRING (20 80, 98

POLYGON((175 150, 20 40, 50 60, 125

190, 110 180, 50 75 )’::geometry)

) As sline;

ST\_GeomFromText(’POINT(110 170)’), 2

)

sline

) As slinewkt;

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

LINESTRING(100 100,73.0769230769231

115.384615384615)

←

LINESTRING(140.752120669087 ←

125.695053378061,121.111404660392 15

**请参考**

ST\_ClosestPoint, ST\_Distance, ST\_LongestLine, ST\_MaxDistance

**8.9.47**

**ST\_Touches**

ST\_Touches — 如果两个几何对象最少有一个共同点且他们的内部并不相交，那么返回TRUE

**用法**

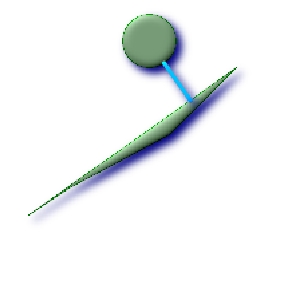
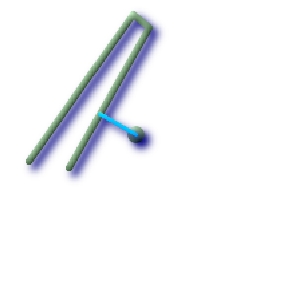
boolean ST\_Touches(geometry g1, geometry g2);

**描述**

如果输入几何对象g1和g2在他们的边界上面只有一个共同点，那么返回TRUE。该函数适用于如下组合：

Area/Area, Line/Line, Line/Area, Point/Area and Point/Line ，但是不适用于g1和g2都是点

用数学公式表示如下：



←

ST\_GeomFromText(’ ←

ST\_Buffer( ←

PostGIS 2.2.0dev 开发手册

288 / 686

两个几何对象允许的DE-9IM规范相交结果矩阵如下：

· FT\*\*\*\*\*\*\*

· F\*\*T\*\*\*\*\*

· F\*\*\*T\*\*\*\*

**重要**

该函数不支持GEOMETRYCOLLECTION类型对象作为输入参数

**注意**

该函数会自动生成一个bounding box以便充分利用几何对象上的任意索引. To avoid using an index, use \_ST\_Touches instead.

该函数方法实现了规范 OpenGIS Simple Features Implementation Speciﬁcation for SQL 1.1. s2.1.1.2 // s2.1.13.3

该函数方法实现了规范 SQL/MM speciﬁcation. SQL-MM 3: 5.1.28

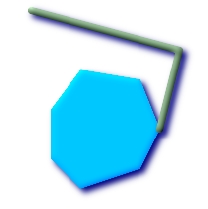
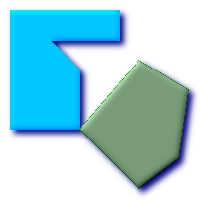
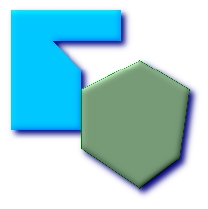
**样例**

如下情形，函数ST\_Touche均返回TRUE

POLYGON / POLYGON

POLYGON / POLYGON

POLYGON / LINESTRING



PostGIS 2.2.0dev 开发手册

289 / 686

LINESTRING / LINESTRING

LINESTRING / LINESTRING

POLYGON / POINT

SELECT ST\_Touches(’LINESTRING(0 0, 1 1, 0 2)’::geometry, ’POINT(1 1)’::geometry);

st\_touches

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

f

(1 row)

SELECT ST\_Touches(’LINESTRING(0 0, 1 1, 0 2)’::geometry, ’POINT(0 2)’::geometry);

st\_touches

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

t

(1 row)

**8.9.48**

**ST\_Within**

ST\_Within — 如果输入几何对象A完全在几何对象B范围内，则返回TRUE

**用法**

boolean ST\_Within(geometry A, geometry B);

**描述**

如果输入几何对象A完全在几何对象B范围内，则返回TRUE，该函数的输入对象必须在相同的坐标投影钟，有相同的SRID才有意义。如果ST\_Within(A,B) 的值是true，同时ST\_Within(B,A) 也是true，那么这两个几何对象是完全相同的

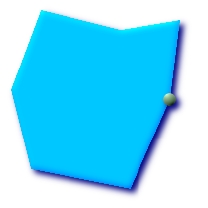
该函数由GEOS模块支持

**重要**

该函数不支持GEOMETRYCOLLECTION类型对象作为输入参数

**重要**

不要将该函数应用于无效的几何对象上，否则你会得到不可预料的结果.



PostGIS 2.2.0dev 开发手册

290 / 686

该函数使用时候会自动生成一个bounding box，以便充分利用几何对象上的任意一个索引。如果想避免使用索引，使用函数\_ST\_Within.这是一个返回逻辑值而不是整型值的可用版本.‘

该函数方法实现了规范 OpenGIS Simple Features Implementation Speciﬁcation for SQL 1.1. s2.1.1.2 // s2.1.13.3 -

a.Relate(b, ’T\*F\*\*F\*\*\*’)

该函数方法实现了规范 SQL/MM speciﬁcation. SQL-MM 3: 5.1.30

**样例**

--a circle within a circle

SELECT ST\_Within(smallc,smallc) As smallinsmall,

ST\_Within(smallc, bigc) As smallinbig,

ST\_Within(bigc,smallc) As biginsmall,

ST\_Within(ST\_Union(smallc, bigc), bigc) as unioninbig,

ST\_Within(bigc, ST\_Union(smallc, bigc)) as biginunion,

ST\_Equals(bigc, ST\_Union(smallc, bigc)) as bigisunion

FROM

(

SELECT ST\_Buffer(ST\_GeomFromText(’POINT(50 50)’), 20) As smallc,

ST\_Buffer(ST\_GeomFromText(’POINT(50 50)’), 40) As bigc) As foo;

--Result

smallinsmall | smallinbig | biginsmall | unioninbig | biginunion | bigisunion

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

t

| t

| f

| t

| t

| t

(1 row)

**请参考**

ST\_Contains, ST\_Equals, ST\_IsValid

**8.10**

**使用SFCGAL 高级2D/3D 函数**

译者注：原始英文版本缺少该库的介绍及安装



PostGIS 2.2.0dev 开发手册

291 / 686

**8.10.1**

**ST\_Extrude**

ST\_Extrude — 把一个Surface类型对象拉伸一定体积。

**用法**

geometry ST\_Extrude(geometry geom, ﬂoat x, ﬂoat y, ﬂoat z);

译者注：该函数原版手册中没有给出样例，请参考：<http://postgis.net/docs/ST_Extrude.html>

**描述**

可用性

该函数方法需要SFCGAL库的支持.

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

该函数支持 Polyhedral Surface类型几何对象.

该函数支持 Triangles 和 Triangulated Irregular Network Surfaces (TIN)类型.

**8.10.2**

**ST\_StraightSkeleton**

ST\_StraightSkeleton — 从一个几何对象中计算出一个直线组成的框架。（译者注：意思就像人体是由一套骨骼组成一样的，该函数的作用就是找出构成这个几何对象的骨架部分）

**用法**

geometry ST\_StraightSkeleton(geometry geom);

译者注：该函数原版手册中没有给出样例，请参考：http://postgis.net/docs/ST\_StraightSkeleton.html

**描述**

可用性

该函数方法需要SFCGAL库的支持.

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

该函数支持 Polyhedral Surface类型几何对象.

该函数支持 Triangles 和 Triangulated Irregular Network Surfaces (TIN)类型.

**8.10.3**

**ST\_IsPlanar**

ST\_IsPlanar —检验一个面是否是平面

**用法**

boolean ST\_IsPlanar(geometry geom);



PostGIS 2.2.0dev 开发手册

292 / 686

**描述**

可用性

该函数方法需要SFCGAL库的支持.

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

该函数支持 Polyhedral Surface类型几何对象.

该函数支持 Triangles 和 Triangulated Irregular Network Surfaces (TIN)类型.

**8.10.4**

**ST\_Orientation**

ST\_Orientation — 确定表面方向

**用法**

integer ST\_Orientation(geometry geom);

**描述**

可用性

该函数方法需要SFCGAL库的支持.

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

该函数支持 Polyhedral Surface类型几何对象.

该函数支持 Triangles 和 Triangulated Irregular Network Surfaces (TIN)类型.

**8.10.5**

**ST\_ForceLHR**

ST\_ForceLHR —强制对输入的几何对象做左手定则转换

**用法**

geometry ST\_ForceLHR(geometry geom);

**描述**

可用性

该函数方法需要SFCGAL库的支持.

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

该函数支持 Polyhedral Surface类型几何对象.

该函数支持 Triangles 和 Triangulated Irregular Network Surfaces (TIN)类型.



PostGIS 2.2.0dev 开发手册

293 / 686

**8.10.6**

**ST\_MinkowskiSum**

ST\_MinkowskiSum — 求 Minkowski （闵可夫斯基）和

译者注：关于闵可夫斯基和定义如下：

闵可夫斯基和是两个欧几里得空间的点集的和，以德国数学家闵可夫斯基命名。点集A与B的闵可夫斯基和就是A+B={a+b|a∈A,b∈B}。例如，平面上有两个三角形，其坐标分别为A = {(1, 0), (0, 1), (0, −1)}及B = {(0, 0), (1, 1), (1, −1)}，则其闵可夫斯基和为A + B = {(1, 0), (2, 1), (2, −1), (0, 1), (1, 2), (1, 0), (0, −1), (1, 0), (1, −2)}。

**用法**

geometry ST\_Minkowski(geometry geom1, geometry geom2);

**描述**

该函数方法需要SFCGAL库的支持.

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

该函数支持 Polyhedral Surface类型几何对象.

该函数支持 Triangles 和 Triangulated Irregular Network Surfaces (TIN)类型.

**8.10.7**

**ST\_Tesselate**

ST\_Tesselate —对一个Surface进行曲面三角形化切割（译者注：该函数的作用就是把各个surface面切成有三角形组成的面）

**用法**

geometry ST\_Tesselate(geometry geom);

**样例**

原文手册没有给出具体的样例，参考官网给出的： http://postgis.net/docs/ST\_Tesselate.html

**描述**

可用性

该函数方法需要SFCGAL库的支持.

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

该函数支持 Polyhedral Surface类型几何对象.

该函数支持 Triangles 和 Triangulated Irregular Network Surfaces (TIN)类型.

**8.11**

**8.11.1**

**几何对象处理函数**

**ST\_Buffer**

ST\_Buffer —对geometry类型对象来说，返回以给定点为中心点，距离小于或等于指定距离的所有点组成的几何对象。计算方式是按照spatial\_ref\_sys表中对该类型的几何对象的描述来计算的。对于geography类型对象：使用一个2D维度转换包装器。PostGIS 1.5版本引入了对不同的end cap和mitre参数的设置来控制几何对象的形状。buffer\_style参数控制如下：

quad\_segs=#,endcap=round|ﬂat|square,join=round|mitre|bevel,mitre\_limit=#.#



PostGIS 2.2.0dev 开发手册

294 / 686

**用法**

geometry ST\_Buffer(geometry g1, ﬂoat radius\_of\_buffer);

geometry ST\_Buffer(geometry g1, ﬂoat radius\_of\_buffer, integer num\_seg\_quarter\_circle);

geometry ST\_Buffer(geometry g1, ﬂoat radius\_of\_buffer, text buffer\_style\_parameters);

geography ST\_Buffer(geography g1, ﬂoat radius\_of\_buffer\_in\_meters);

**描述**

对geometry类型对象来说，返回以给定点为中心点，距离小于或等于指定距离的所有点组成的几何对象。计算方式是按照spatial\_ref\_sys表中对该类型的几何对象的描述来计算的。对于geography类型对象：使用一个2D维度转换包装器。PostGIS 1.5版本引入了对不同的end cap和mitre参数的设置来控制几何对象的形状。

**注意**

负值半径：对于polygon对象来说，使用负值的半径，这意味着收缩polygon，而不是扩展它。

**注意**

geography类型对象:对geography类型对象的来说，该函数实际上是对geometry类型实现的简单包装。它首先会选择一个适用于geography 对象的bounding box的最佳SRID值。(偏向于使用参考系UTM, Lambert Azimuthal Equal Area (LAEA)，南北极，最差的情况是使用Mercator墨卡托投影) ，然后在平面空间参考系中做缓冲，最后再重新转换到WGS 84地理参考系中

对于geography类型对象来说，如果输入对象大到跨了两个UTM十区或者穿过了国际日期变更线，那么输出结果可能和预想的有不同

可用版本: 1.5 – 这次2.2.0版本ST\_Buffer提升了对不同的end cap 和join类型（译者注：这两个专业术语应该是不同几何对象的结合方式，这里给出原文）的支持。在如下场景很有用，例如把LINESTRING类型对象转换成polygon类型对象，LINESTRING之间结合方式是直角而不是圆角。对于geography类型对象的简单函数包装也支持

需要版本 GEOS >= 3.2 来充分利用geometry功能

第三个参数（目前只支持geometry类型）：可以指定一个分隔四分之一圆的弧段的数量（默认四分之圆做7次分隔，即8个弧段）或者空格分隔键值对参数：

’quad\_segs=#’ : 用来近似四分之一圆的弧段数，默认是8

’endcap=round|ﬂat|square’ : endcap 方式 (默认是"round", 需要 GEOS-3.2 或更高版本支持). ’butt’ 和“flat” 意义等价，也可以是参数值

’join=round|mitre|bevel’ : join 方式 (默认是"round", 需要 GEOS-3.2 或更高版本支持). ’miter’ 和’mitre’意义等价，也可以是参数值

’mitre\_limit=#.#’ : 切割率限制 (只影响miter 方式的join). ’miter\_limit’ 和’mitre\_limit’ 意义等价，也可以是参数值

半径单位以空间参考系规定的单位为准

输入参数可以是POINTS, MULTIPOINTS, LINESTRINGS, MULTILINESTRINGS, POLYGONS, MULTIPOLYGONS, 和GeometryCollections等类型

**注意**

该函数不考虑第三维度(z坐标)，即便输入对象是一个3D几何对象，返回的也是2D的buffer



PostGIS 2.2.0dev 开发手册

295 / 686

该函数由GEOS模块支持.

该函数方法实现了规范 OpenGIS Simple Features Implementation Speciﬁcation for SQL 1.1. s2.1.1.3

该函数方法实现了规范 SQL/MM speciﬁcation. SQL-MM 3: 5.1.17

**注意**

用户经常错误性地使用该函数来做半径搜索。创建一个缓冲区来做半径搜索是很慢且无效的。应该使用函数ST\_DWithin来处理.

**样例**

quad\_segs=8 (default)

SELECT ST\_Buffer(

ST\_GeomFromText(’POINT(100 90)’),

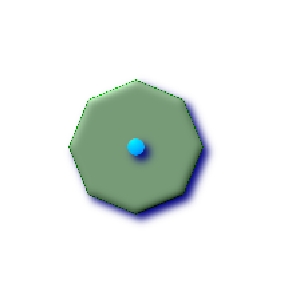
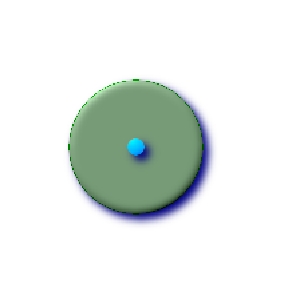
50, ’quad\_segs=8’);

quad\_segs=2 (lame)

SELECT ST\_Buffer(

ST\_GeomFromText(’POINT(100 90)’),

50, ’quad\_segs=2’);



PostGIS 2.2.0dev 开发手册

296 / 686

endcap=round join=round (default)

SELECT ST\_Buffer(

ST\_GeomFromText(

’LINESTRING(50 50,150 150,150 50)’

), 10, ’endcap=round join=round’);

join=bevel

SELECT ST\_Buffer(

ST\_GeomFromText(

’LINESTRING(50 50,150 150,150 50)’

), 10, ’join=bevel’);

--A buffered point approximates a circle

endcap=square

SELECT ST\_Buffer(

ST\_GeomFromText(

’LINESTRING(50 50,150 150,150 50)’

), 10, ’endcap=square join=round’);

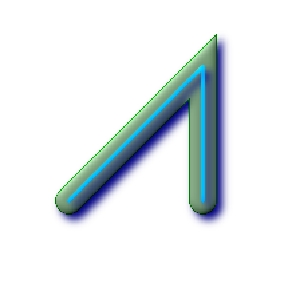
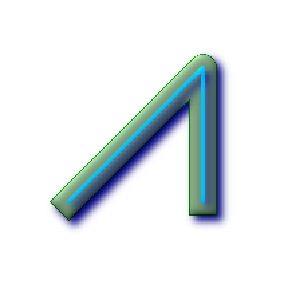
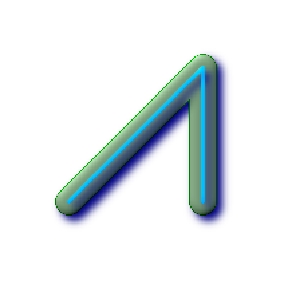
join=mitre mitre\_limit=5.0 (default mitre limit)

SELECT ST\_Buffer(

ST\_GeomFromText(

’LINESTRING(50 50,150 150,150 50)’

), 10, ’join=mitre mitre\_limit=5.0’);



PostGIS 2.2.0dev 开发手册

297 / 686

-- A buffered point forcing approximation of (see diagram)

-- 2 points per circle is poly with 8 sides (see diagram)

SELECT ST\_NPoints(ST\_Buffer(ST\_GeomFromText(’POINT(100 90)’), 50)) As

←

promisingcircle\_pcount,

ST\_NPoints(ST\_Buffer(ST\_GeomFromText(’POINT(100 90)’), 50, 2)) As lamecircle\_pcount;

promisingcircle\_pcount | lamecircle\_pcount

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

33 | 9

--A lighter but lamer circle

-- only 2 points per quarter circle is an octagon

--Below is a 100 meter octagon

-- 注意 coordinates are in NAD 83 long lat which we transform

to Mass state plane meter and then buffer to get measurements in meters;

SELECT ST\_AsText(ST\_Buffer(

ST\_Transform(

ST\_SetSRID(ST\_MakePoint(-71.063526, 42.35785),4269), 26986)

,100,2)) As octagon;

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

POLYGON((236057.59057465 900908.759918696,236028.301252769 900838.049240578,235

957.59057465 900808.759918696,235886.879896532 900838.049240578,235857.59057465

900908.759918696,235886.879896532 900979.470596815,235957.59057465 901008.759918

696,236028.301252769 900979.470596815,236057.59057465 900908.759918696))

**请参考**

ST\_Collect, ST\_DWithin, ST\_SetSRID, ST\_Transform, ST\_Union

**8.11.2**

**ST\_BuildArea**

ST\_BuildArea —根据给定的几何对象，创建一个由线条类对象组成的平面几何对象

**用法**

geometry ST\_BuildArea(geometry A);

**描述**

根据给定的几何对象，创建一个由线条组成的平面几何对象。根据输入类型，返回的类型可以是Polygon 或 MultiPolygon。如果输入的线条不能组成polygon，那么返回NULL。输入的线条类对象类型可以是LINESTRINGS,MULTILINESTRINGS, POLYGONS, MULTIPOLYGONS, 和 GeometryCollections.

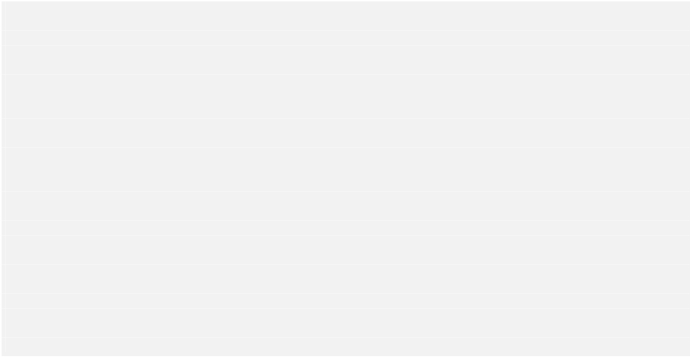
该函数假定所有的输入几何类型对象都有空

**注意**

输入的线条类对象必须是由正确的点组成，才能使用该函数

可用版本: 1.1.0 -需要GEOS版本>= 2.1.0.

**样例**



PostGIS 2.2.0dev 开发手册

298 / 686

This will create a donut

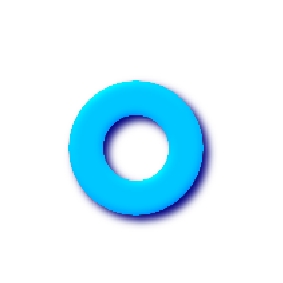
SELECT ST\_BuildArea(ST\_Collect(smallc,bigc))

FROM (SELECT

ST\_Buffer(

ST\_GeomFromText(’POINT(100 90)’), 25) As smallc,

ST\_Buffer(ST\_GeomFromText(’POINT(100 90)’), 50) As bigc) As foo;



PostGIS 2.2.0dev 开发手册

299 / 686

This will create a gaping hole inside the circle with prongs sticking out

SELECT ST\_BuildArea(ST\_Collect(line,circle))

FROM (SELECT

ST\_Buffer(

ST\_MakeLine(ST\_MakePoint(10, 10),ST\_MakePoint(190, 190)),

5)

As line,

ST\_Buffer(ST\_GeomFromText(’POINT(100 90)’), 50) As circle) As foo;

--this creates the same gaping hole

--but using linestrings instead of polygons

SELECT ST\_BuildArea(

ST\_Collect(ST\_ExteriorRing(line),ST\_ExteriorRing(circle))

)

FROM (SELECT ST\_Buffer(

ST\_MakeLine(ST\_MakePoint(10, 10),ST\_MakePoint(190, 190))

,5)

As line,

ST\_Buffer(ST\_GeomFromText(’POINT(100 90)’), 50) As circle) As foo;

**请参考**

ST\_Node, ST\_MakePolygon, ST\_BdPolyFromText, ST\_BdMPolyFromText 这些函数使用OGC 接口对该函数进行包装

**8.11.3**

**ST\_Collect**

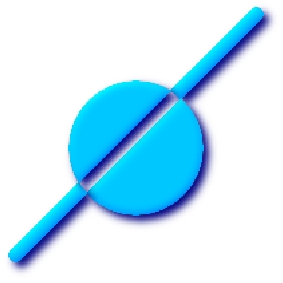
ST\_Collect — 从其他几何类型对象的collection 返回一个具体的ST\_Geometry值（对应的对象）

**用法**

geometry ST\_Collect(geometry set g1ﬁeld);

geometry ST\_Collect(geometry g1, geometry g2);

geometry ST\_Collect(geometry[] g1\_array);



PostGIS 2.2.0dev 开发手册

300 / 686

**描述**

输出类型可以是MULTI\* 或GEOMETRYCOLLECTION类型。该函数有两种形式，形式1输入参数是两个geometry类型对象，形式2是一个聚合功能函数，输入对象是一个几何类型对象的集合，返回一个ST\_Geometry函数值对应的几何对象。

聚合版本，即形式2：根据一个几何类型对象的集合，该函数返回一个GEOMETRYCOLLECTION 或MULTI 类型对象。在PostgreSQL术语中，函数ST\_Collect()是一个“聚合”（译者注：所谓聚合函数就是类似count，sum这样的函数）。聚合函数意味着，对于多行的数据，该函数的作用方式会像SQL里面的聚合函数SUM() and AVG() 一样。例如"SELECT ST\_Collect(GEOM) FROM GEOMTABLE GROUP BY ATTRCOLUMN" 会返回一个按ATTRCOLUMN列聚合的GEOMETRYCOLLECTION类型。

非聚合函数版本该函数会根据输入的两个几何对象，返回一个collection对象，输出类型可能是MULTI\* 或 GEOMETRYCOLLECTION类型

**注意**

函数ST\_Collect 和函数 ST\_Union 经常可以互换使用。函数ST\_Collect 一般来说要比函数ST\_Union 快很多，因为ST\_Collect函数不会去分解输入几何对象的边界或者检查一个MultiPolgon对象是否有重叠部分。它仅仅是把单个的geometry类型对象组合成一个MULTI类型对象，或者将MULTI类型对象组合成一个Geometry Collections集合对象。不幸的是，GIS工具通常不能很好地支持geometry collection类型。为了阻止函数ST\_Collect 把MULTI类型对象变成Geometry Collection 类型对象，用户可以使用如下小技巧：利用函数ST\_Dump把MULTI类型对象转换成单个geometry类型对象，再重新聚合他们。

可用版本: 1.4.0 - 引入函数ST\_Collect(geomarray)支持. ST\_Collect函数用于更快地处理更多的几何对象

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

该函数支持CircularString和Curve几何类型对象，但该函数不会像你期待的那样返回一个 MULTICURVE 类型或MULTI类型对象，PostGIS目前还不支持这些

**样例**

Aggregate example [(http://postgis.refractions.net/pipermail/postgis-users/2008-June/020331.html)](http://postgis.refractions.net/pipermail/postgis-users/2008-June/020331.html)

SELECT stusps,

ST\_Multi(ST\_Collect(f.the\_geom)) as singlegeom

FROM (SELECT stusps, (ST\_Dump(the\_geom)).geom As the\_geom

FROM

somestatetable ) As f

GROUP BY stusps

Non-Aggregate example

SELECT ST\_AsText(ST\_Collect(ST\_GeomFromText(’POINT(1 2)’),

ST\_GeomFromText(’POINT(-2 3)’) ));

st\_astext

----------

MULTIPOINT(1 2,-2 3)

--Collect 2 d points

SELECT ST\_AsText(ST\_Collect(ST\_GeomFromText(’POINT(1 2)’),

ST\_GeomFromText(’POINT(1 2)’) ) );

st\_astext

----------

MULTIPOINT(1 2,1 2)

--Collect 3d points



PostGIS 2.2.0dev 开发手册

301 / 686

SELECT ST\_AsEWKT(ST\_Collect(ST\_GeomFromEWKT(’POINT(1 2 3)’),

ST\_GeomFromEWKT(’POINT(1 2 4)’) ) );

st\_asewkt

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

MULTIPOINT(1 2 3,1 2 4)

--Example with curves

SELECT ST\_AsText(ST\_Collect(ST\_GeomFromText(’CIRCULARSTRING(220268 150415,220227

←

150505,220227 150406)’),

ST\_GeomFromText(’CIRCULARSTRING(220227 150406,2220227 150407,220227 150406)’)));

st\_astext

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

GEOMETRYCOLLECTION(CIRCULARSTRING(220268 150415,220227 150505,220227 150406),

CIRCULARSTRING(220227 150406,2220227 150407,220227 150406))

--New ST\_Collect array construct

SELECT ST\_Collect(ARRAY(SELECT the\_geom FROM sometable));

SELECT ST\_AsText(ST\_Collect(ARRAY[ST\_GeomFromText(’LINESTRING(1 2, 3 4)’),

ST\_GeomFromText(’LINESTRING(3 4, 4 5)’)])) As wktcollect;

--wkt collect --

MULTILINESTRING((1 2,3 4),(3 4,4 5))

**请参考**

ST\_Dump, ST\_Union

**8.11.4**

**ST\_ConcaveHull**

ST\_ConcaveHull — 返回包含所有输入几何对象的凹包。你可以认为这是一个收缩的包

**用法**

geometry ST\_ConcaveHull(geometry geomA, ﬂoat target\_percent, boolean allow\_holes=false);

**描述**

返回包含所有输入几何对象的凹包。你可以认为这是一个收缩的包。默认不允许带孔（洞洞）的polygon对象。返回的几何对象的维度不会高于单例polygon对象。

参数target\_percent，PostGIS在不断地进行包收缩结束前，所能得到的凹包面积的占凸包面积的比例值。

译者注：该参数较难理解，可以这么理解，首先有一个初始的凸包，然后不断地进行收缩，target\_percent就相当于“压缩比”，但这个压缩比并不能一定能达到，有可能在达到指定的目标值前因为引发了拓扑异常而使得该函数停止收缩，并返回最终异常发生前的凹包

你可以把凹包看成对一个几何对象进行“真空压缩”后的结果。（译者注：就像超市里面卖的真空包装酒鬼花生一样，在打开前，包装壳就相当于一个凹包）。参数target\_percent值为1的时候，得到结果和凸包一样。当该参数的值在0到0.99之间时，你会得到一个面积比凸包小一些的结果。该函数与凸包不同，凸包就像在几何对象集合外面放一个橡皮圈一样。（译者注：打开酒鬼花生，空气进入里面，这时候就像一个凸包一样~~）

该函数通常用于MULTI 和Geometry Collection类型对象。虽然该函数不是聚合函数，但你可以与函数ST\_Collect 或 ST\_Union 一起使用来获得一个Point/LINESTRING/POLYGON对象集合的凹包，例如ST\_ConcaveHull(ST\_Collect(somepointﬁeld),0.80).

该函数计算凹包比计算凸包要慢很多，但是会更好地包围几何对象，在图像识别领域也很有用

该函数由GEOS模块支持



PostGIS 2.2.0dev 开发手册

302 / 686

**注意**

注意 –如果输入对象是Point类型、LINESTRING类型或者geometry collection 类型对象，使用函数ST\_Collect，如果你的处理对象是Polygon类型对象，使用函数ST\_Union，因为对于无效的geometry类型对象，ST\_ConcaveHull会失败

**注意**

注意 –参数target\_percent的值越小，生成凹包的所需要的时间越长，也越可能导致拓扑异常。同样也会获得更多的单精度点。首先尝试该参数第一阶值为0.99，通常会非常快，有时候和计算凸包一样快，通常收缩超过99%，因为该函数总是会超出收缩一些。然后设置改值0.98，通常会更慢，其他值也会更慢。为了减小精度和得到的单精度点的数量，使用函数ST\_ConcaveHull后再使用函数ST\_SimplifyPreserveTopology 或 ST\_SnapToGrid 。函数ST\_SnapToGrid稍微快一些，但会导致无效的机会对象，而函数ST\_SimplifyPreserveTopology却总是会验证输入几何对象的有效性。

更多真实世界的例子和技术介绍参考文档<http://www.bostongis.com/postgis_concavehull.snippet>

同样参考Simon Greener关于介绍Oracle 11G R2的凹包的文章http://www.spatialdbadvisor.com/[oracle\_spatial\_tips\_tricks/172/concave-hull-geometries-in-oracle-11gr2.](http://www.spatialdbadvisor.com/oracle_spatial_tips_tricks/172/concave-hull-geometries-in-oracle-11gr2)

我们使用target\_percent值为0.75获得了与Simon使用Oracle函数SDO\_CONCAVEHULL\_BOUNDARY得到类似的凹包形状

可用版本: 2.0.0

**样例**

--Get estimate of infected area based on point observations

SELECT d.disease\_type,

ST\_ConcaveHull(ST\_Collect(d.pnt\_geom), 0.99) As geom

FROM disease\_obs As d

GROUP BY d.disease\_type;



PostGIS 2.2.0dev 开发手册

303 / 686

ST\_ConcaveHull of 2 polygons encased in target 100%

shrink concave hull

-- geometries overlaid with concavehull

-- at target 100% shrink (this is the ←

same as convex hull - since no shrink)

SELECT

ST\_ConcaveHull(

ST\_Union(ST\_GeomFromText ←

(’POLYGON((175 150, 20 40,

50 60, 125 100, ←

175 150))’),

ST\_Buffer(ST\_GeomFromText ←

(’POINT(110 170)’), 20)

), 1)

As convexhull;

-- geometries overlaid with concavehull at target 90% of

convex hull area

-- geometries overlaid with concavehull ←

at target 90% shrink

SELECT

ST\_ConcaveHull(

ST\_Union(ST\_GeomFromText ←

(’POLYGON((175 150, 20 40,

50 60, 125 100, ←

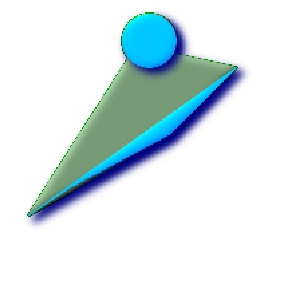
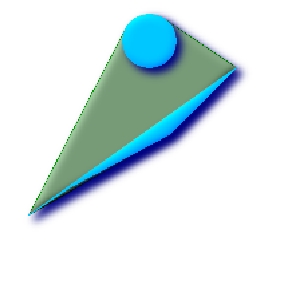
175 150))’),

ST\_Buffer(ST\_GeomFromText ←

(’POINT(110 170)’), 20)

), 0.9)

As target\_90;



PostGIS 2.2.0dev 开发手册

304 / 686

L Shape points overlaid with convex hull

-- this produces a table of 42 points

that form an L shape

SELECT (ST\_DumpPoints(ST\_GeomFromText(

’MULTIPOINT(14 14,34 14,54 14,74 14,94

←

←

14,114 14,134 14,

150 14,154 14,154 6,134 6,114 6,94 6,74

←

ST\_ConcaveHull of L points at target 99% of convex hull

6,54 6,34 6,

14 6,10 6,8 6,7 7,6 8,6 10,6 30,6 50,6

70,6 90,6 110,6 130,

6 150,6 170,6 190,6 194,14 194,14 174,14

←

←

SELECT ST\_ConcaveHull(ST\_Collect(geom),

0.99)

FROM l\_shape;

←

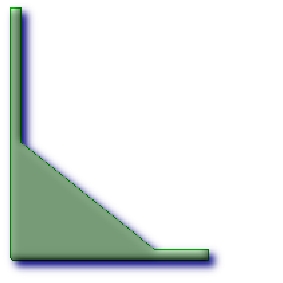
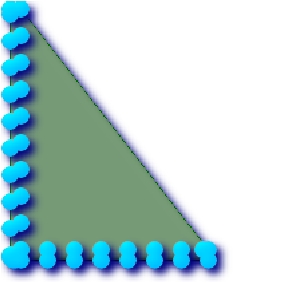
154,14 134,14 114,

14 94,14 74,14 54,14 34,14 14)’))).geom

INTO TABLE l\_shape;

SELECT ST\_ConvexHull(ST\_Collect(geom))

FROM l\_shape;



PostGIS 2.2.0dev 开发手册

305 / 686

multilinestring overlaid with Convex hull

Concave Hull of L points at target 80% convex hull area

-- Concave Hull L shape points

-- at target 80% of convexhull

SELECT ST\_ConcaveHull(ST\_Collect( ←

geom), 0.80)

FROM l\_shape;

multilinestring with overlaid with Concave hull of

linestrings at 99% target -- ﬁrst hop

SELECT ST\_ConcaveHull(ST\_GeomFromText(’ ←

MULTILINESTRING((106 164,30 112,74 70,82 112,1

130 62,122 40,156 32,162 76,172

88),

←

(132 178,134 148,128 136,96 128,132

←

108,150 130,

170 142,174 110,156 96,158 90,158 88),

(22 64,66 28,94 38,94 68,114 76,112 30,

132 10,168 18,178 34,186 52,184 74,190

100,

←

190 122,182 148,178 170,176 184,156

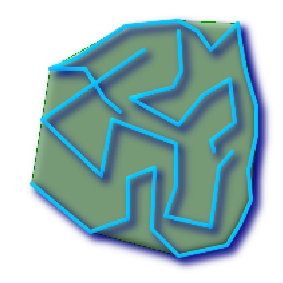
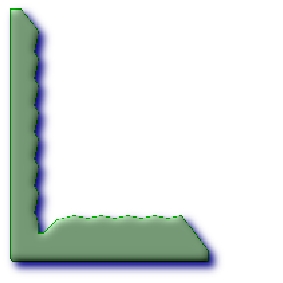
164,146 178,

←

132 186,92 182,56 158,36 150,62 150,76

128,88 118))’),0.99)

←



PostGIS 2.2.0dev 开发手册

306 / 686

**请参考**

ST\_Collect, ST\_ConvexHull, ST\_SimplifyPreserveTopology, ST\_SnapToGrid

**8.11.5**

**ST\_ConvexHull**

ST\_ConvexHull — 返回包含所有几何对象的最小的凸包

**用法**

geometry ST\_ConvexHull(geometry geomA);

**描述**

返回包含所有几何对象的最小的凸包

你可以把凸包看成包围一个几何对象的一个弹性的带子。该函数与凹包不同。该函数通常应用于处理MULTI 和 Geometry Collections类型对象。虽然该函数不是一个聚合函数，但你可以用函数ST\_Collect来获取一个点集的凸包如下： ST\_ConvexHull(ST\_Collect(somepointﬁeld)).

它经常被用来在一组观测点上面确定一个受影响的区域（译者注：比如病毒传播啦，在一些点上面生成一个可能受影响的面）

该函数由GEOS模块支持

该函数方法实现了规范 OpenGIS Simple Features Implementation Speciﬁcation for SQL 1.1. s2.1.1.3

该函数方法实现了规范 SQL/MM speciﬁcation. SQL-MM 3: 5.1.16

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

**样例**

--Get estimate of infected area based on point observations

SELECT d.disease\_type,

ST\_ConvexHull(ST\_Collect(d.the\_geom)) As the\_geom

FROM disease\_obs As d

GROUP BY d.disease\_type;



PostGIS 2.2.0dev 开发手册

307 / 686

Convex Hull of a MultiLinestring and a MultiPoint seen together with the MultiLinestring and MultiPoint

SELECT ST\_AsText(ST\_ConvexHull(

ST\_Collect(

ST\_GeomFromText(’MULTILINESTRING((100 190,10 8),(150 10, 20 30))’),

ST\_GeomFromText(’MULTIPOINT(50 5, 150 30, 50 10, 10 10)’)

)) );

---st\_astext--

POLYGON((50 5,10 8,10 10,100 190,150 30,150 10,50 5))

**请参考**

ST\_Collect, ST\_ConcaveHull, ST\_MinimumBoundingCircle

**8.11.6**

**ST\_CurveToLine**

ST\_CurveToLine — 把一个CIRCULARSTRING/CURVEDPOLYGON对象转换成一个LINESTRING/POLYGON对象

**用法**

geometry ST\_CurveToLine(geometry curveGeom);

geometry ST\_CurveToLine(geometry curveGeom, integer segments\_per\_qtr\_circle);

**描述**

把一个CIRCULARSTRING对象转换成一个规则的LINESTRING对象，或者把一个CURVEDPOLYGON对象转换成一个规则的POLYGON对象。对于那些不支持CIRCULARSTRING几何类型的平台上，这很有用。

把一个输入的机会对象转换成一个线性的几何对象。每一个Curve几何对象或者片段会被分隔转换成近似线段的处理，默认分隔处理方法是每四分之一圆分成32弧段。

可用版本: 1.2.2?

该函数方法实现了规范 [OpenGIS Simple Features Implementation Speciﬁcation for SQL 1.1.](http://www.opengeospatial.org/standards/sfs)

该函数方法实现了规范 SQL/MM speciﬁcation. SQL-MM 3: 7.1.7



PostGIS 2.2.0dev 开发手册

308 / 686

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

该函数支持CircularString和Curve几何类型对象

**样例**

SELECT ST\_AsText(ST\_CurveToLine(ST\_GeomFromText(’CIRCULARSTRING(220268 150415,220227

150505,220227 150406)’)));

--Result --

←

LINESTRING(220268 150415,220269.95064912 150416.539364228,220271.823415575

150418.17258804,220273.613787707 150419.895736857,

←

220275.317452352 150421.704659462,220276.930305234 150423.594998003,220278.448460847

150425.562198489,

220279.868261823 150427.60152176,220281.186287736 150429.708054909,220282.399363347

150431.876723113,

220283.50456625 150434.10230186,220284.499233914 150436.379429536,220285.380970099

150438.702620341,220286.147650624 150441.066277505,

220286.797428488 150443.464706771,220287.328738321 150445.892130112,220287.740300149

150448.342699654,

220288.031122486 150450.810511759,220288.200504713 150453.289621251,220288.248038775

150455.77405574,

220288.173610157 150458.257830005,220287.977398166 150460.734960415,220287.659875492

150463.199479347,

220287.221807076 150465.64544956,220286.664248262 150468.066978495,220285.988542259

150470.458232479,220285.196316903 150472.81345077,

220284.289480732 150475.126959442,220283.270218395 150477.39318505,220282.140985384

150479.606668057,

←

←

←

←

←

←

←

←

220280.90450212 150481.762075989,220279.5637474 150483.85421628,220278.12195122

150485.87804878,

←

220276.582586992 150487.828697901,220274.949363179 150489.701464356,220273.226214362

150491.491836488,

220271.417291757 150493.195501133,220269.526953216 150494.808354014,220267.559752731

150496.326509628,

220265.520429459 150497.746310603,220263.41389631 150499.064336517,220261.245228106

150500.277412127,

220259.019649359 150501.38261503,220256.742521683 150502.377282695,220254.419330878

150503.259018879,

220252.055673714 150504.025699404,220249.657244448 150504.675477269,220247.229821107

150505.206787101,

220244.779251566 150505.61834893,220242.311439461 150505.909171266,220239.832329968

150506.078553494,

220237.347895479 150506.126087555,220234.864121215 150506.051658938,220232.386990804

150505.855446946,

220229.922471872 150505.537924272,220227.47650166 150505.099855856,220225.054972724

150504.542297043,

220222.663718741 150503.86659104,220220.308500449 150503.074365683,

220217.994991777 150502.167529512,220215.72876617 150501.148267175,

220213.515283163 150500.019034164,220211.35987523 150498.7825509,

220209.267734939 150497.441796181,220207.243902439 150496,

220205.293253319 150494.460635772,220203.420486864 150492.82741196,220201.630114732

150491.104263143,

220199.926450087 150489.295340538,220198.313597205 150487.405001997,220196.795441592

150485.437801511,

220195.375640616 150483.39847824,220194.057614703 150481.291945091,220192.844539092

150479.123276887,220191.739336189 150476.89769814,

220190.744668525 150474.620570464,220189.86293234 150472.297379659,220189.096251815

150469.933722495,

←

←

←

←

←

←

←

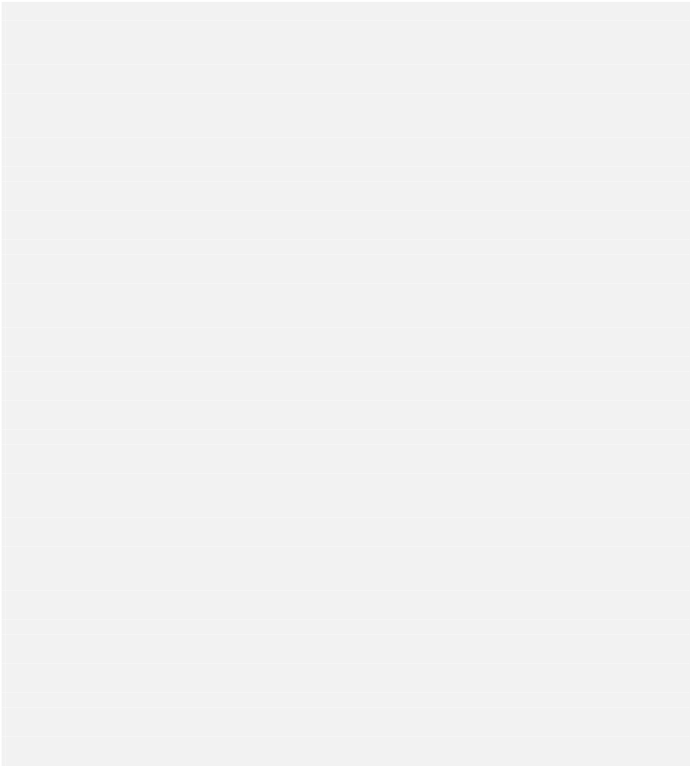
←

←

←

←

←



PostGIS 2.2.0dev 开发手册

309 / 686

220188.446473951 150467.535293229,220187.915164118 150465.107869888,220187.50360229

150462.657300346,

220187.212779953 150460.189488241,220187.043397726 150457.710378749,220186.995863664

150455.22594426,

220187.070292282 150452.742169995,220187.266504273 150450.265039585,220187.584026947

150447.800520653,

220188.022095363 150445.35455044,220188.579654177 150442.933021505,220189.25536018

150440.541767521,

220190.047585536 150438.18654923,220190.954421707 150435.873040558,220191.973684044

150433.60681495,

220193.102917055 150431.393331943,220194.339400319 150429.237924011,220195.680155039

150427.14578372,220197.12195122 150425.12195122,

220198.661315447 150423.171302099,220200.29453926 150421.298535644,220202.017688077

150419.508163512,220203.826610682 150417.804498867,

220205.716949223 150416.191645986,220207.684149708 150414.673490372,220209.72347298

150413.253689397,220211.830006129 150411.935663483,

220213.998674333 150410.722587873,220216.22425308 150409.61738497,220218.501380756

150408.622717305,220220.824571561 150407.740981121,

220223.188228725 150406.974300596,220225.586657991 150406.324522731,220227 150406)

--3d example

←

←

←

←

←

←

←

←

←

SELECT ST\_AsEWKT(ST\_CurveToLine(ST\_GeomFromEWKT(’CIRCULARSTRING(220268 150415 1,220227

150505 2,220227 150406 3)’)));

Output

------

LINESTRING(220268 150415 1,220269.95064912 150416.539364228 1.0181172856673,

220271.823415575 150418.17258804 1.03623457133459,220273.613787707 150419.895736857

1.05435185700189,....AD INFINITUM ....

220225.586657991 150406.324522731 1.32611114201132,220227 150406 3)

--use only 2 segments to approximate quarter circle

SELECT ST\_AsText(ST\_CurveToLine(ST\_GeomFromText(’CIRCULARSTRING(220268 150415,220227

150505,220227 150406)’),2));

st\_astext

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

←

←

←

LINESTRING(220268 150415,220287.740300149 150448.342699654,220278.12195122

←

150485.87804878,

220244.779251566 150505.61834893,220207.243902439 150496,220187.50360229 150462.657300346,

220197.12195122 150425.12195122,220227 150406)

**请参考**

ST\_LineToCurve

**8.11.7**

**ST\_DelaunayTriangles**

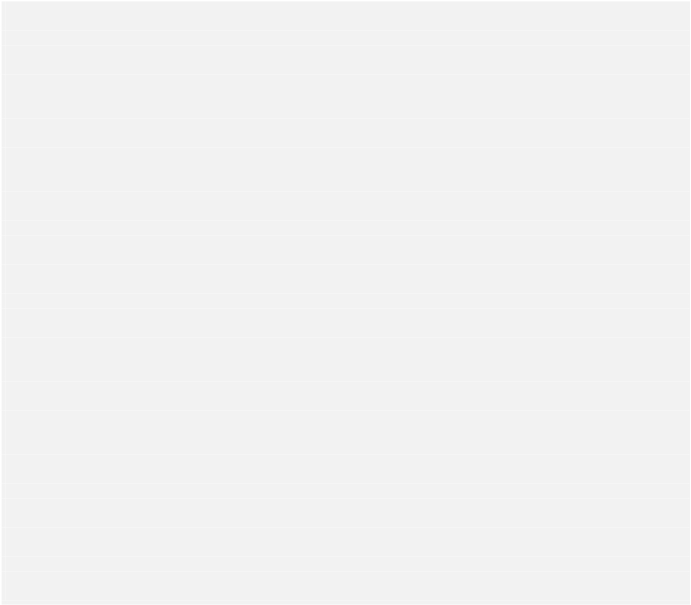
ST\_DelaunayTriangles —根据输入对象的顶点对象，返回一个Delaunay 三角形化的结果。（译者注：所谓三角形化就是把输入的几何对象分成一个个三角形，有点类似微积分里面的思想）

**用法**

geometry ST\_DelaunayTriangles(geometry g1, ﬂoat tolerance, int4 ﬂags);

**描述**

根据输入对象的顶点对象，返回一个Delaunay 三角形化的结果。如果参数flags为0，则返回的是polygon对象的COLLECTION ，如果flags值为1，返回MULTILINESTRING ，如果flags值为2，则返回TIN。如果参数tolerance没有值，则函数会把输入对象变成一些顶点



PostGIS 2.2.0dev 开发手册

310 / 686

可用版本: 2.1.0 -需要GEOS版本>= 3.4.0.

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

该函数支持 Triangles 和 Triangulated Irregular Network Surfaces (TIN)类型.

**2D 样例**

Original polygons

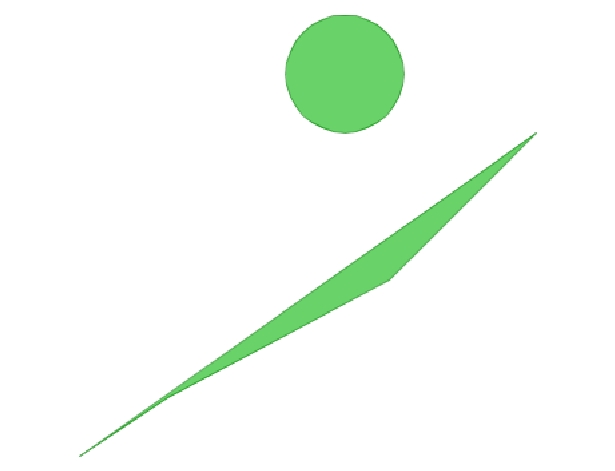
-- our original geometry --

ST\_Union(ST\_GeomFromText(’POLYGON((175 150, 20 40,

50 60, 125 100, 175 150))’),

ST\_Buffer(ST\_GeomFromText(’POINT(110 170)’), 20)

)



PostGIS 2.2.0dev 开发手册

311 / 686

ST\_DelaunayTriangles of 2 polygons: delaunay triangle polygons each triangle themed in different color

-- geometries overlaid multilinestring triangles

SELECT

ST\_DelaunayTriangles(

ST\_Union(ST\_GeomFromText(’POLYGON((175 150, 20 40,

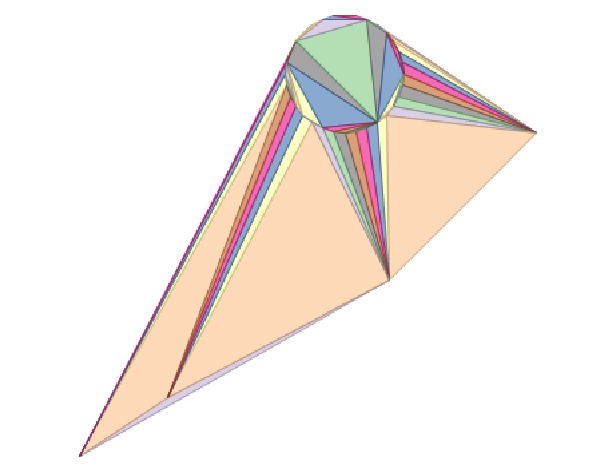
50 60, 125 100, 175 150))’),

ST\_Buffer(ST\_GeomFromText(’POINT(110 170)’), 20)

))

As

dtriag;



PostGIS 2.2.0dev 开发手册

312 / 686

-- delaunay triangles as multilinestring

SELECT

ST\_DelaunayTriangles(

ST\_Union(ST\_GeomFromText(’POLYGON((175 150, 20 40,

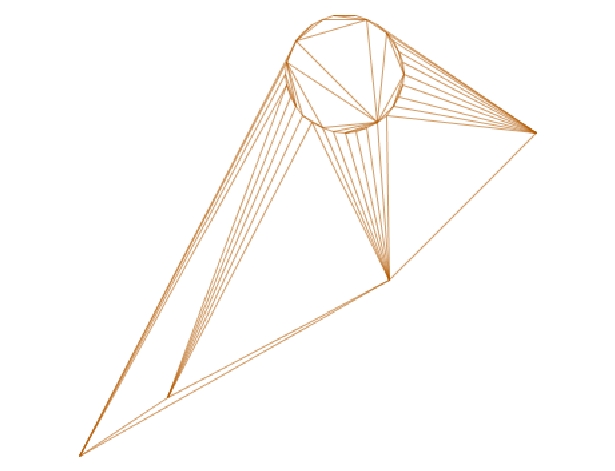
50 60, 125 100, 175 150))’),

ST\_Buffer(ST\_GeomFromText(’POINT(110 170)’), 20)

),0.001,1)

As

dtriag;



PostGIS 2.2.0dev 开发手册

313 / 686

-- delaunay triangles of 45 points as 55 triangle polygons

-- this produces a table of 42 points that form an L shape

SELECT (ST\_DumpPoints(ST\_GeomFromText(

’MULTIPOINT(14 14,34 14,54 14,74 14,94 14,114 14,134 14,

150 14,154 14,154 6,134 6,114 6,94 6,74 6,54 6,34 6,

14 6,10 6,8 6,7 7,6 8,6 10,6 30,6 50,6 70,6 90,6 110,6 130,

6 150,6 170,6 190,6 194,14 194,14 174,14 154,14 134,14 114,

14 94,14 74,14 54,14 34,14 14)’))).geom

INTO TABLE l\_shape;

-- output as individual polygon triangles

SELECT ST\_AsText((ST\_Dump(geom)).geom) As wkt

FROM ( SELECT ST\_DelaunayTriangles(ST\_Collect(geom)) As geom

FROM l\_shape) As foo;

---wkt ---

POLYGON((6 194,6 190,14 194,6 194))

POLYGON((14 194,6 190,14 174,14 194))

POLYGON((14 194,14 174,154 14,14 194))

POLYGON((154 14,14 174,14 154,154 14))

POLYGON((154 14,14 154,150 14,154 14))

POLYGON((154 14,150 14,154 6,154 14))

:

:

**3D 样例**

-- 3D multipoint --

SELECT ST\_AsText(ST\_DelaunayTriangles(ST\_GeomFromText(

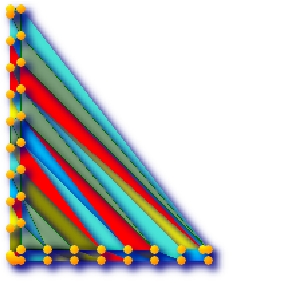
’MULTIPOINT Z(14 14 10,

150 14 100,34 6 25, 20 10 150)’))) As wkt;

-----wkt----

GEOMETRYCOLLECTION Z (POLYGON Z ((14 14 10,20 10 150,34 6 25,14 14 10))

,POLYGON Z ((14 14 10,34 6 25,150 14 100,14 14 10)))



PostGIS 2.2.0dev 开发手册

314 / 686

**请参考**

ST\_ConcaveHull, ST\_Dump

**8.11.8**

**ST\_Difference**

ST\_Difference — 根据输入的几何对象A和B，返回一个几何对象A的除了和几何对象B相交的剩余部分

**用法**

geometry ST\_Difference(geometry geomA, geometry geomB);

**描述**

根据输入的几何对象A和B，返回一个几何对象A的除了和几何对象B相交的剩余部分。你可以这样理解：

GeometryA - ST\_Intersection(A,B)。如果A完全被B包含，那么返回一个空几何类型

**注意**

注意 – 该函数的返回值与输入对象的顺序有关， B - A 返回的是除了A∩B后，B中剩余的部分

该函数由GEOS模块支持

**注意**

该函数不支持GeometryCollection参数

该函数方法实现了规范 OpenGIS Simple Features Implementation Speciﬁcation for SQL 1.1. s2.1.1.3

该函数方法实现了规范 SQL/MM speciﬁcation. SQL-MM 3: 5.1.20

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）。然而似乎该函数处理时候只考虑x，y坐标，但返回时候会将原有的Z坐标加上

**样例**



PostGIS 2.2.0dev 开发手册

315 / 686

The original linestrings shown together.

The difference of the two linestrings

--Safe for 2d. This is same geometries as what is shown for st\_symdifference

SELECT ST\_AsText(

ST\_Difference(

ST\_GeomFromText(’LINESTRING(50 100, 50 200)’),

ST\_GeomFromText(’LINESTRING(50 50, 50 150)’)

)

);

st\_astext

---------

LINESTRING(50 150,50 200)

--When used in 3d doesn’t quite do the right thing

SELECT ST\_AsEWKT(ST\_Difference(ST\_GeomFromEWKT(’MULTIPOINT(-118.58 38.38 5,-118.60 38.329

6,-118.614 38.281 7)’), ST\_GeomFromEWKT(’POINT(-118.614 38.281 5)’)));

st\_asewkt

---------

MULTIPOINT(-118.6 38.329 6,-118.58 38.38 5)

**请参考**

ST\_SymDifference

←

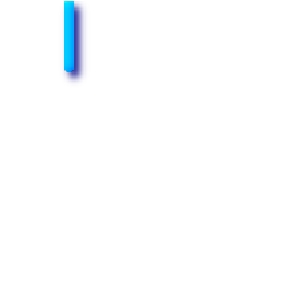
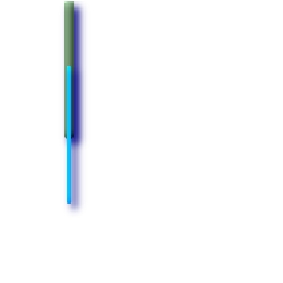
**8.11.9**

**ST\_Dump**

ST\_Dump —返回一个组成输入几何对象g1的geometry\_dump (geom,path) 行集合

**用法**

geometry\_dump[] ST\_Dump(geometry g1);



PostGIS 2.2.0dev 开发手册

316 / 686

**描述**

这是一个返回集合的函数（set-returning function 即SRF）。该函数会返回组成输入对象的geometry\_dump 行集合和行的位置值。当输入类型是 (POINT,LINESTRING,POLYGON)，返回的是一个空的数组和一个和输入对象一样的几何对象形成的一条记录。当输入对象是一个collection或者multi类型的collection，该函数会返回每一个collection的组成和组成每一个collection的每一个元素的位置下标。

ST\_Dump 在展开geometry对象很有用，它与GROUP BY子句的功能是相反的。该函数可以用来把MULTIPOLYGONS 展开成 POLYGONS.

版本提升: 2.0.0 版本支持Polyhedral Surface类型、Triangle类型、TIN类型.

可用版本: PostGIS 1.0.0RC1. Requires PostgreSQL 7.3 or higher.

**注意**

1.3.4版本前,该函数如果包含CURVE类型对象会crash掉，1.3.4+版本均已修复

该函数支持CircularString和Curve几何类型对象

该函数支持 Polyhedral Surface类型几何对象.

该函数支持 Triangles 和 Triangulated Irregular Network Surfaces (TIN)类型.

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

**Standard 样例**

SELECT sometable.field1, sometable.field1,

(ST\_Dump(sometable.the\_geom)).geom AS the\_geom

FROM sometable;

-- Break a compound curve into its constituent linestrings and circularstrings

SELECT ST\_AsEWKT(a.geom), ST\_HasArc(a.geom)

FROM ( SELECT (ST\_Dump(p\_geom)).geom AS geom

FROM (SELECT ST\_GeomFromEWKT(’COMPOUNDCURVE(CIRCULARSTRING(0 0, 1 1, 1 0),(1 0, 0

1))’) AS p\_geom) AS b

) AS a;

←

st\_asewkt

| st\_hasarc

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

CIRCULARSTRING(0 0,1 1,1 0) | t

LINESTRING(1 0,0 1) | f

(2 rows)

**Polyhedral Surfaces, TIN and Triangle 样例**

-- Polyhedral surface example

-- Break a Polyhedral surface into its faces

SELECT (a.p\_geom).path[1] As path, ST\_AsEWKT((a.p\_geom).geom) As geom\_ewkt

FROM (SELECT ST\_Dump(ST\_GeomFromEWKT(’POLYHEDRALSURFACE(

((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)),

((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),

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

((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1))

((1 1 0, 1 1 ←



PostGIS 2.2.0dev 开发手册

317 / 686

)’) ) AS p\_geom )

path |

AS a;

geom\_ewkt

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

1 | POLYGON((0 0 0,0 0 1,0 1 1,0 1 0,0 0 0))

2 | POLYGON((0 0 0,0 1 0,1 1 0,1 0 0,0 0 0))

3 | POLYGON((0 0 0,1 0 0,1 0 1,0 0 1,0 0 0))

4 | POLYGON((1 1 0,1 1 1,1 0 1,1 0 0,1 1 0))

5 | POLYGON((0 1 0,0 1 1,1 1 1,1 1 0,0 1 0))

6 | POLYGON((0 0 1,1 0 1,1 1 1,0 1 1,0 0 1))

-- TIN --

SELECT (g.gdump).path, ST\_AsEWKT((g.gdump).geom) as wkt

FROM

(SELECT

ST\_Dump( ST\_GeomFromEWKT(’TIN (((

0 0 0,

0 0 1,

0 1 0,

0 0 0

)), ((

0 0 0,

0 1 0,

1 1 0,

0 0 0

))

)’) ) AS gdump

) AS g;

-- result --

path |

wkt

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

{1} | TRIANGLE((0 0 0,0 0 1,0 1 0,0 0 0))

{2} | TRIANGLE((0 0 0,0 1 0,1 1 0,0 0 0))

**请参考**

geometry\_dump, Section 13.5, ST\_Collect, ST\_Collect, ST\_GeometryN

**8.11.10**

**ST\_DumpPoints**

ST\_DumpPoints —返回组成一个几何对象的所有点。

**用法**

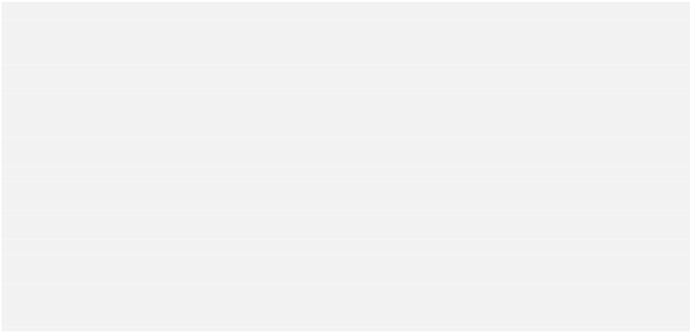
geometry\_dump[]ST\_DumpPoints(geometry geom);

**描述**

该函数会返回组成输入对象的geometry\_dump 行集合和行的位置值。

输入对象是由点组成的几何对象。

参数path是一个展开输入几何对象的后的所有点的位置形成的数组，每一个点的位置就是该数组的下标。例如如果输入对象是一个LINESTRING对象，会返回一个LINESTRING所有点位置坐标的数组{i}。如果输入对象是一个POLYGON，返回一个数组{i,j}，i表示环的位置顺序（1是最外面的环，内环以此类推），j是环上面的点的位置顺序（下标从1开始）



PostGIS 2.2.0dev 开发手册

318 / 686

版本提升: 2.1.0 速度更快，用C重新实现了该函数

版本提升: 2.0.0 版本支持Polyhedral Surface类型、Triangle类型、TIN类型.

可用版本: 1.5.0

该函数支持CircularString和Curve几何类型对象

该函数支持 Polyhedral Surface类型几何对象.

该函数支持 Triangles 和 Triangulated Irregular Network Surfaces (TIN)类型.

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

**Classic Explode a Table of LineStrings into nodes**

SELECT edge\_id, (dp).path[1] As index, ST\_AsText((dp).geom) As wktnode

FROM (SELECT 1 As edge\_id

, ST\_DumpPoints(ST\_GeomFromText(’LINESTRING(1 2, 3 4, 10 10)’)) AS dp

UNION ALL

SELECT 2 As edge\_id

, ST\_DumpPoints(ST\_GeomFromText(’LINESTRING(3 5, 5 6, 9 10)’)) AS dp

) As foo;

edge\_id | index |

wktnode

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

1 | 1 | POINT(1 2)

1 | 2 | POINT(3 4)

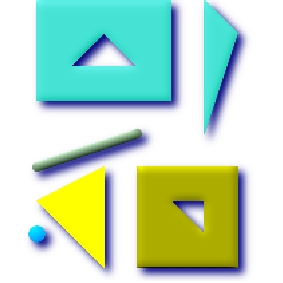
1 | 3 | POINT(10 10)

2 | 1 | POINT(3 5)

2 | 2 | POINT(5 6)

2 | 3 | POINT(9 10)

**Standard Geometry 样例**



PostGIS 2.2.0dev 开发手册

319 / 686

SELECT path, ST\_AsText(geom)

FROM (

SELECT (ST\_DumpPoints(g.geom)).\*

FROM

(SELECT

’GEOMETRYCOLLECTION(

POINT ( 0 1 ),

LINESTRING ( 0 3, 3 4 ),

POLYGON (( 2 0, 2 3, 0 2,

POLYGON (( 3 0, 3 3, 6 3,

( 5 1, 4 2, 5 2,

MULTIPOLYGON (

(( 0 5, 0 8, 4 8,

( 1 6, 3 6, 2 7,

(( 5 4, 5 8, 6 7,

)

)’::geometry AS geom

) AS g

) j;

2 0 )),

6 0, 3 0 ),

5 1 )),

4 5, 0 5 ),

1 6 )),

5 4 ))

path

| st\_astext

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

{1,1} | POINT(0 1)

{2,1} | POINT(0 3)

{2,2} | POINT(3 4)

{3,1,1} | POINT(2 0)

{3,1,2} | POINT(2 3)

{3,1,3} | POINT(0 2)

{3,1,4} | POINT(2 0)

{4,1,1} | POINT(3 0)

{4,1,2} | POINT(3 3)

{4,1,3} | POINT(6 3)

{4,1,4} | POINT(6 0)

{4,1,5} | POINT(3 0)

{4,2,1} | POINT(5 1)

{4,2,2} | POINT(4 2)

{4,2,3} | POINT(5 2)

{4,2,4} | POINT(5 1)

{5,1,1,1} | POINT(0 5)

{5,1,1,2} | POINT(0 8)

{5,1,1,3} | POINT(4 8)

{5,1,1,4} | POINT(4 5)

{5,1,1,5} | POINT(0 5)

{5,1,2,1} | POINT(1 6)

{5,1,2,2} | POINT(3 6)

{5,1,2,3} | POINT(2 7)

{5,1,2,4} | POINT(1 6)

{5,2,1,1} | POINT(5 4)

{5,2,1,2} | POINT(5 8)

{5,2,1,3} | POINT(6 7)

{5,2,1,4} | POINT(5 4)

(29 rows)

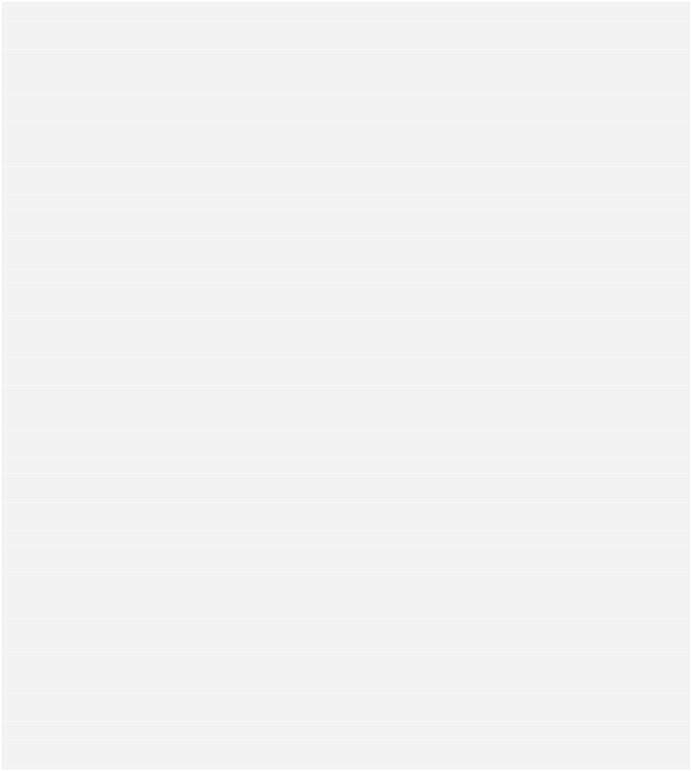
**Polyhedral Surfaces, TIN and Triangle 样例**

-- Polyhedral surface cube --

SELECT (g.gdump).path, ST\_AsEWKT((g.gdump).geom) as wkt

FROM

(SELECT



PostGIS 2.2.0dev 开发手册

320 / 686

ST\_DumpPoints(ST\_GeomFromEWKT(’POLYHEDRALSURFACE( ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0

0)),

((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),

((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),

((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)) )’) ) AS gdump

) AS g;

-- result --

←

path

|

wkt

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

{1,1,1} | POINT(0 0 0)

{1,1,2} | POINT(0 0 1)

{1,1,3} | POINT(0 1 1)

{1,1,4} | POINT(0 1 0)

{1,1,5} | POINT(0 0 0)

{2,1,1} | POINT(0 0 0)

{2,1,2} | POINT(0 1 0)

{2,1,3} | POINT(1 1 0)

{2,1,4} | POINT(1 0 0)

{2,1,5} | POINT(0 0 0)

{3,1,1} | POINT(0 0 0)

{3,1,2} | POINT(1 0 0)

{3,1,3} | POINT(1 0 1)

{3,1,4} | POINT(0 0 1)

{3,1,5} | POINT(0 0 0)

{4,1,1} | POINT(1 1 0)

{4,1,2} | POINT(1 1 1)

{4,1,3} | POINT(1 0 1)

{4,1,4} | POINT(1 0 0)

{4,1,5} | POINT(1 1 0)

{5,1,1} | POINT(0 1 0)

{5,1,2} | POINT(0 1 1)

{5,1,3} | POINT(1 1 1)

{5,1,4} | POINT(1 1 0)

{5,1,5} | POINT(0 1 0)

{6,1,1} | POINT(0 0 1)

{6,1,2} | POINT(1 0 1)

{6,1,3} | POINT(1 1 1)

{6,1,4} | POINT(0 1 1)

{6,1,5} | POINT(0 0 1)

(30 rows)

-- Triangle --

SELECT (g.gdump).path, ST\_AsText((g.gdump).geom) as wkt

FROM

(SELECT

ST\_DumpPoints( ST\_GeomFromEWKT(’TRIANGLE ((

0 0,

0 9,

9 0,

0 0

))’) ) AS gdump

) AS g;

-- result --

path |

wkt

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

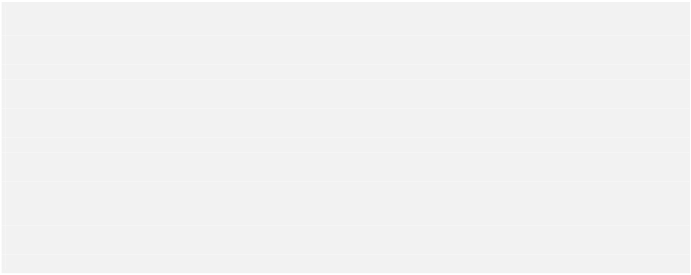
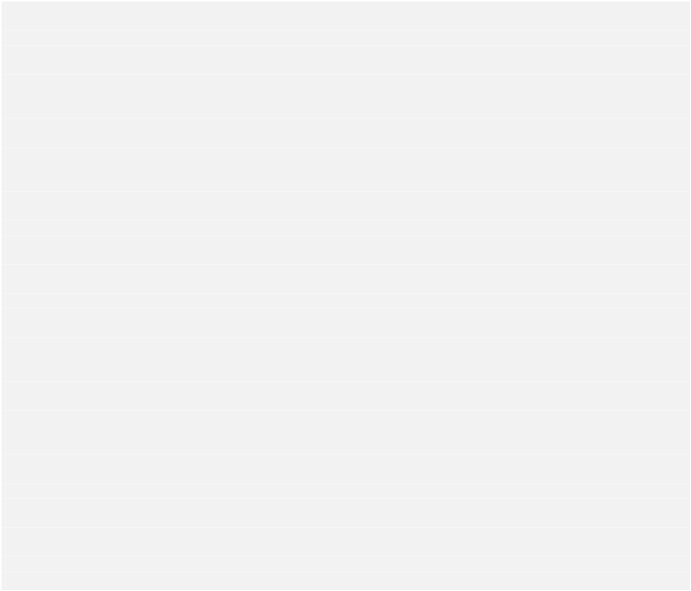
{1} | POINT(0 0)

{2} | POINT(0 9)

{3} | POINT(9 0)

{4} | POINT(0 0)

-- TIN --



PostGIS 2.2.0dev 开发手册

321 / 686

SELECT (g.gdump).path, ST\_AsEWKT((g.gdump).geom) as wkt

FROM

(SELECT

ST\_DumpPoints( ST\_GeomFromEWKT(’TIN (((

0 0 0,

0 0 1,

0 1 0,

0 0 0

)), ((

0 0 0,

0 1 0,

1 1 0,

0 0 0

))

)’) ) AS gdump

) AS g;

-- result --

path

|

wkt

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

{1,1,1} | POINT(0 0 0)

{1,1,2} | POINT(0 0 1)

{1,1,3} | POINT(0 1 0)

{1,1,4} | POINT(0 0 0)

{2,1,1} | POINT(0 0 0)

{2,1,2} | POINT(0 1 0)

{2,1,3} | POINT(1 1 0)

{2,1,4} | POINT(0 0 0)

(8 rows)

**请参考**

geometry\_dump, Section 13.5, ST\_Dump, ST\_DumpRings

**8.11.11**

**ST\_DumpRings**

ST\_DumpRings — 返回将一个polygon对象 展开后dump出外环和内环的结果

**用法**

geometry\_dump[] ST\_DumpRings(geometry a\_polygon);

**描述**

返回将一个polygon对象 展开后dump出外环和内环的结果

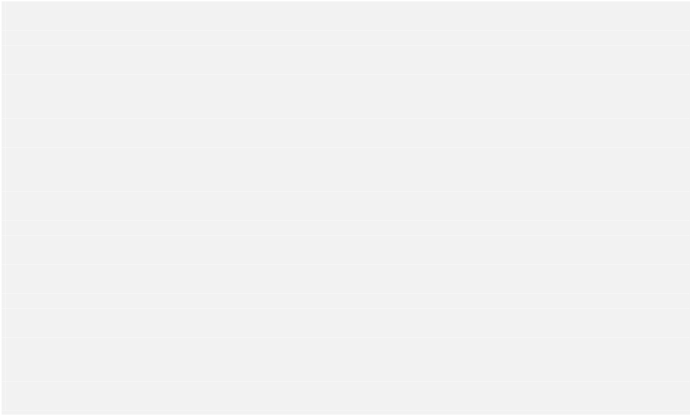
参数"path" 表示环的标号，0表示外环，大于0表示内环的标号。输入对象是一个polygon对象

可用版本: PostGIS 1.1.3. Requires PostgreSQL 7.3 or higher.

**注意**

该函数目前只支持POLYGON几何类型对象，不支持MULTIPOLYGON类型对象

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.



PostGIS 2.2.0dev 开发手册

322 / 686

**样例**

SELECT sometable.field1, sometable.field1,

(ST\_DumpRings(sometable.the\_geom)).geom As the\_geom

FROM sometableOfpolys;

SELECT ST\_AsEWKT(geom) As the\_geom, path

FROM ST\_DumpRings(

ST\_GeomFromEWKT(’POLYGON((-8149064 5133092 1,-8149064 5132986 1,-8148996 5132839

1,-8148972 5132767 1,-8148958 5132508 1,-8148941 5132466 1,-8148924 5132394 1,

←

-8148903 5132210 1,-8148930 5131967 1,-8148992 5131978 1,-8149237 5132093 1,-8149404

5132211 1,-8149647 5132310 1,-8149757 5132394 1,

-8150305 5132788 1,-8149064 5133092 1),

(-8149362 5132394 1,-8149446 5132501 1,-8149548 5132597 1,-8149695 5132675 1,-8149362

5132394 1))’)

←

←

)

as foo;

path |

the\_geom

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

{0} | POLYGON((-8149064 5133092 1,-8149064 5132986 1,-8148996 5132839 1,-8148972 5132767

1,-8148958 5132508 1,

| -8148941 5132466 1,-8148924 5132394 1,

| -8148903 5132210 1,-8148930 5131967 1,

| -8148992 5131978 1,-8149237 5132093 1,

| -8149404 5132211 1,-8149647 5132310 1,-8149757 5132394 1,-8150305 5132788

1,-8149064 5133092 1))

{1} | POLYGON((-8149362 5132394 1,-8149446 5132501 1,

| -8149548 5132597 1,-8149695 5132675 1,-8149362 5132394 1))

**请参考**

geometry\_dump, Section 13.5, ST\_Dump, ST\_ExteriorRing, ST\_InteriorRingN

←

←

**8.11.12**

**ST\_FlipCoordinates**

ST\_FlipCoordinates —互换一个输入几何对象的X和Y坐标，对于用户将经度/维度坐标错误输入成维度/经度时候很有用，可以用这个函数来修复它

**用法**

geometry ST\_FlipCoordinates(geometry geom);

**描述**

互换一个输入几何对象的X和Y坐标.

该函数支持CircularString和Curve几何类型对象

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

该函数支持M测量值坐标.

可用版本: 2.0.0



PostGIS 2.2.0dev 开发手册

323 / 686

该函数支持 Polyhedral Surface类型几何对象.

该函数支持 Triangles 和 Triangulated Irregular Network Surfaces (TIN)类型.

**Example**

SELECT ST\_AsEWKT(ST\_FlipCoordinates(GeomFromEWKT(’POINT(1 2)’)));

st\_asewkt

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

POINT(2 1)

**8.11.13**

**ST\_Intersection**

ST\_Intersection —返回几何对象相交的点集对应的geometry对象。对于geography类型对象，该函数会先将其转化成geometry类型对象，然后再转换成WGS84下的坐标

**用法**

geometry ST\_Intersection( geometry geomA , geometry geomB );

geography ST\_Intersection( geography geogA , geography geogB );

**描述**

返回几何对象相交的点集对应的geometry对象

换句话说，返回的几何对象是输入的两个几何对象的公共部分

如果输入的几何对象没有任何公共的部分或者不相交，那么该函数返回一个空对象

ST\_Intersection 函数通常和ST\_Intersect函数一起使用，比如当你只想得到你感兴趣的国家或区域的部分点，可以用该函数来缩小涉及到 bounding box, buffer, region这种查询中的范围

**注意**

对于geography类型对象：该函数只是对geometry 实现的简单包装。该函数首先将其转换成geometry对象，然后在平面参考系中相交得到交集，然后转换回WGS84参考系，变成geography类型对象

**重要**

该函数不支持GEOMETRYCOLLECTION类型对象作为输入参数

该函数由GEOS模块支持

该方法由SFCGAL库提供后台支持.

可用版本: 1.5 引入对geography数据类型的支持.

该函数方法实现了规范 OpenGIS Simple Features Implementation Speciﬁcation for SQL 1.1. s2.1.1.3

该函数方法实现了规范 SQL/MM speciﬁcation. SQL-MM 3: 5.1.18



PostGIS 2.2.0dev 开发手册

324 / 686

**样例**

SELECT ST\_AsText(ST\_Intersection(’POINT(0 0)’::geometry, ’LINESTRING ( 2 0, 0 2 )’:: ←

geometry));

st\_astext

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

GEOMETRYCOLLECTION EMPTY

(1 row)

SELECT ST\_AsText(ST\_Intersection(’POINT(0 0)’::geometry, ’LINESTRING ( 0 0, 0 2 )’:: ←

geometry));

st\_astext

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

POINT(0 0)

(1 row)

---Clip all lines (trails) by country (here we assume country geom are POLYGON or

MULTIPOLYGONS)

←

-- NOTE: we are only keeping intersections that result in a LINESTRING or MULTILINESTRING

because we don’t

-- care about trails that just share a point

-- the dump is needed to expand a geometry collection into individual single MULT\* parts

-- the below is fairly generic and will work for polys, etc. by just changing the where

←

←

clause

SELECT clipped.gid, clipped.f\_name, clipped\_geom

FROM (SELECT trails.gid, trails.f\_name, (ST\_Dump(ST\_Intersection(country.the\_geom, trails. ←

the\_geom))).geom As clipped\_geom

FROM country

INNER JOIN trails

ON ST\_Intersects(country.the\_geom, trails.the\_geom))

WHERE ST\_Dimension(clipped.clipped\_geom) = 1 ;

As clipped

--For polys e.g. polygon landmarks, you can also use the sometimes faster hack that

buffering anything by 0.0

-- except a polygon results in an empty geometry collection

--(so a geometry collection containing polys, lines and points)

-- buffered by 0.0 would only leave the polygons and dissolve the collection shell

←

SELECT poly.gid,

ST\_Multi(ST\_Buffer(

ST\_Intersection(country.the\_geom, poly.the\_geom),

0.0)

) As clipped\_geom

FROM country

INNER JOIN poly

ON ST\_Intersects(country.the\_geom, poly.the\_geom)

WHERE Not ST\_IsEmpty(ST\_Buffer(ST\_Intersection(country.the\_geom, poly.the\_geom),0.0));

**请参考**

ST\_Difference, ST\_Dimension, ST\_Dump, ST\_SymDifference, ST\_Intersects, ST\_Multi

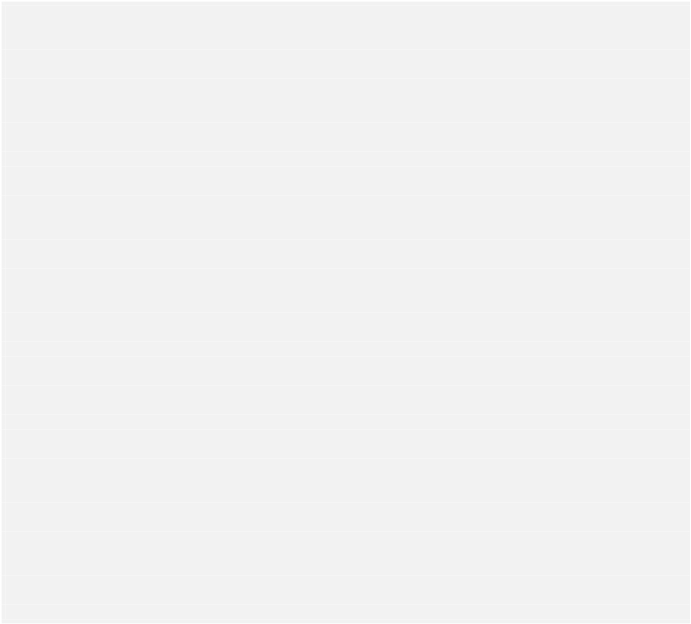
**8.11.14**

**ST\_LineToCurve**

ST\_LineToCurve — 把一个LINESTRING/POLYGON 转换成一个CIRCULARSTRING, CURVED POLYGON类型对象

**用法**

geometry ST\_LineToCurve(geometry geomANoncircular);



PostGIS 2.2.0dev 开发手册

325 / 686

**描述**

把一个LINESTRING/POLYGON 转换成一个CIRCULARSTRING, CURVED POLYGON类型对象。注意：该函数的作用是，转换后的几何对象所需要的点来等价地表示Curve类型对象

可用版本: 1.2.2?

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

该函数支持CircularString和Curve几何类型对象

**样例**

SELECT ST\_AsText(ST\_LineToCurve(foo.the\_geom)) As curvedastext,ST\_AsText(foo.the\_geom) As

non\_curvedastext

FROM (SELECT ST\_Buffer(’POINT(1 3)’::geometry, 3) As the\_geom) As foo;

←

curvedatext

non\_curvedastext

--------------------------------------------------------------------|--------------------------------

CURVEPOLYGON(CIRCULARSTRING(4 3,3.12132034355964 0.878679656440359, | POLYGON((4

3,3.94235584120969 2.41472903395162,3.77163859753386 1.85194970290473,

←

1 0,-1.12132034355965 5.12132034355963,4 3)) | 3.49440883690764

1.33328930094119,3.12132034355964 0.878679656440359,

| 2.66671069905881

←

←

0.505591163092366,2.148050297

0.228361402466141,

| 1.58527096604839

←

0.0576441587903094,1 ←

0,

| 0.414729033951621

←

0.0576441587903077,-0.1480502

0.228361402466137,

| -0.666710699058802

←

0.505591163092361,-1.12132034

0.878679656440353,

| -1.49440883690763

←

1.33328930094119,-1.771638597

1.85194970290472

| --ETC--

←

,3.94235584120969

3.58527096604839,4

3))

--3D example

SELECT ST\_AsEWKT(ST\_LineToCurve(ST\_GeomFromEWKT(’LINESTRING(1 2 3, 3 4 8, 5 6 4, 7 8 4, 9

10 4)’)));

st\_asewkt

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

CIRCULARSTRING(1 2 3,5 6 4,9 10 4)

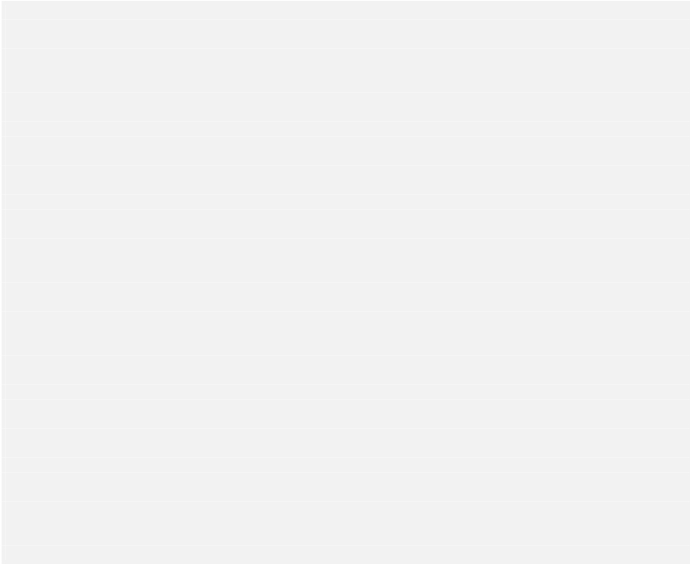
**请参考**

ST\_CurveToLine

←

←

←



PostGIS 2.2.0dev 开发手册

326 / 686

**8.11.15**

**ST\_MakeValid**

ST\_MakeValid —不丢弃顶点的情况下，尝试把无效的几何对象转换成有效的几何对象

**用法**

geometry ST\_MakeValid(geometry input);

**描述**

不丢弃顶点的情况下，尝试把无效的几何对象转换成有效的几何对象

已经是有效的几何对象，则不处理，直接返回。

支持的输入类型有: POINTS, MULTIPOINTS, LINESTRINGS, MULTILINESTRINGS, POLYGONS, MULTIPOLYGONS

和 GEOMETRYCOLLECTIONS 即前面类型的任意混合类型

如果有部分或者全部维度损失，输出的几何对象是一个更低维度几何对象的集合或者一个更低维度的几何对象

有自相交的简单多边形可能会返回成MULTI-几何对象

可用版本: 2.0.0, 需要版本 GEOS-3.3.0支持

版本提升: 2.0.1,提升了性能，需要版本GEOS-3.3.4支持

版本提升: 2.1.0 添加了对GEOMETRYCOLLECTION 和 MULTIPOINT.的支持

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

**请参考**

ST\_IsValid ST\_CollectionExtract

**8.11.16**

**ST\_MemUnion**

ST\_MemUnion — 与函数ST\_Union功能一样，但是会使用更少的内存以及更多的处理时间

**用法**

geometry ST\_MemUnion(geometry set geomﬁeld);

**描述**

**注意**

与函数ST\_Union功能一样，但是会使用更少的内存以及更多的处理时间。这是一个把所有几何对象一次性union起来的聚合函数。与之前的ST\_Union聚合结果相比，后者会先创建一个几何对象数组，然后再union他们

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.



PostGIS 2.2.0dev 开发手册

327 / 686

**样例**

See ST\_Union

**请参考**

ST\_Union

**8.11.17**

**ST\_MinimumBoundingCircle**

ST\_MinimumBoundingCircle —返回包含一个几何对象的最小的Circle Polygon对象。默认对四分之一圆做48等分弧段分割

**用法**

geometry ST\_MinimumBoundingCircle(geometry geomA, integer num\_segs\_per\_qt\_circ=48);

**描述**

返回一个能完全包含一个几何对象的最小的圆形 Polygon对象

**注意**

通过每个弧段是（四分之一圆的48分之一）的基本粒度对polygon进行切割，返回最接近的Circle对象。参数的值可以提高到更大以便获得更精确的结果但这样做会有一些性能损失

该函数通常用于处理MULTI 和 Geometry Collections类型对象。虽然该函数不是一个聚合函数，但你可以用函数ST\_Collect来获取一个几何对象集合的最小的bounding circle，即ST\_MinimumBoundingCircle(ST\_Collect(somepointﬁeld)).

多边形的面积与最小边界圆的面积之比，通常被称为Roeck测试

可用版本: 1.4.0 – 需要 GEOS库支持

**样例**

SELECT d.disease\_type,

ST\_MinimumBoundingCircle(ST\_Collect(d.the\_geom)) As the\_geom

FROM disease\_obs As d

GROUP BY d.disease\_type;



PostGIS 2.2.0dev 开发手册

328 / 686

Minimum bounding circle of a point and linestring. Using 8 segs to approximate a quarter circle

SELECT ST\_AsText(ST\_MinimumBoundingCircle(

ST\_Collect(

ST\_GeomFromEWKT(’LINESTRING(55 75,125 150)’),

ST\_Point(20, 80)), 8

)) As wktmbc;

wktmbc

-----------

POLYGON((135.59714732062 115,134.384753327498 102.690357210921,130.79416296937

←

90.8537670908995,124.963360620072 79.9451031602111,117.116420743937

70.3835792560632,107.554896839789 62.5366393799277,96.6462329091006

←

←

56.70583703063,84.8096427890789 53.115246672502,72.5000000000001

←

51.9028526793802,60.1903572109213 53.1152466725019,48.3537670908996

56.7058370306299,37.4451031602112 62.5366393799276,27.8835792560632

←

←

70.383579256063,20.0366393799278 79.9451031602109,14.20583703063

←

90.8537670908993,10.615246672502 102.690357210921,9.40285267938019 115,10.6152466725019

←

127.309642789079,14.2058370306299 139.1462329091,20.0366393799275

150.054896839789,27.883579256063 159.616420743937,

←

37.4451031602108 167.463360620072,48.3537670908992 173.29416296937,60.190357210921

176.884753327498,

72.4999999999998 178.09714732062,84.8096427890786 176.884753327498,96.6462329091003

173.29416296937,107.554896839789 167.463360620072,

117.116420743937 159.616420743937,124.963360620072 150.054896839789,130.79416296937

139.146232909101,134.384753327498 127.309642789079,135.59714732062 115))

**请参考**

ST\_Collect, ST\_ConvexHull

←

←

←

**8.11.18**

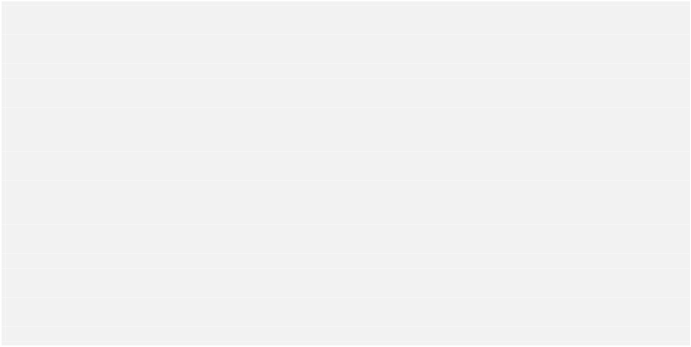
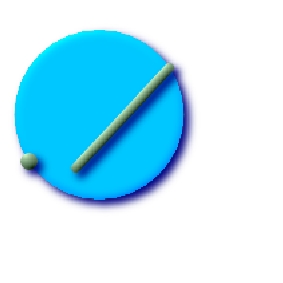
**ST\_Polygonize**

ST\_Polygonize — 聚合函数，从一个线条类的几何类型对象几何创建一个包含Polygon对象的GeometryCollection类型对象

**用法**

geometry ST\_Polygonize(geometry set geomﬁeld);

geometry ST\_Polygonize(geometry[] geom\_array);



PostGIS 2.2.0dev 开发手册

329 / 686

**描述**

聚合函数，从一个线条类的几何类型对象几何创建一个包含Polygon对象的GeometryCollection类型对象

**注意**

第三方工具通常不支持Geometry Collection类型对象，因此使用函数ST\_Polygonize和函数ST\_Dump 把collection 类型的polygon 展开成单例的polygon对象

**注意**

输入的线条类几何对象必须被正确的添加节点分隔才能使用该函数

可用版本: 1.0.0RC1 -需要GEOS版本>= 2.1.0.

**样例: Polygonizing single linestrings**

SELECT ST\_AsEWKT(ST\_Polygonize(the\_geom\_4269)) As geomtextrep

FROM (SELECT the\_geom\_4269 FROM ma.suffolk\_edges ORDER BY tlid LIMIT 45) As foo;

geomtextrep

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

SRID=4269;GEOMETRYCOLLECTION(POLYGON((-71.040878 42.285678,-71.040943 42.2856,-71.04096

42.285752,-71.040878 42.285678)),

←

POLYGON((-71.17166 42.353675,-71.172026 42.354044,-71.17239 42.354358,-71.171794

←

42.354971,-71.170511 42.354855,

-71.17112 42.354238,-71.17166 42.353675)))

(1 row)

--Use ST\_Dump to dump out the polygonize geoms into individual polygons

SELECT ST\_AsEWKT((ST\_Dump(foofoo.polycoll)).geom) As geomtextrep

FROM (SELECT ST\_Polygonize(the\_geom\_4269) As polycoll

FROM (SELECT the\_geom\_4269 FROM ma.suffolk\_edges

ORDER BY tlid LIMIT 45) As foo) As foofoo;

geomtextrep

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

SRID=4269;POLYGON((-71.040878 42.285678,-71.040943 42.2856,-71.04096 42.285752,

-71.040878 42.285678))

SRID=4269;POLYGON((-71.17166 42.353675,-71.172026 42.354044,-71.17239 42.354358

,-71.171794 42.354971,-71.170511 42.354855,-71.17112 42.354238,-71.17166 42.353675))

(2 rows)

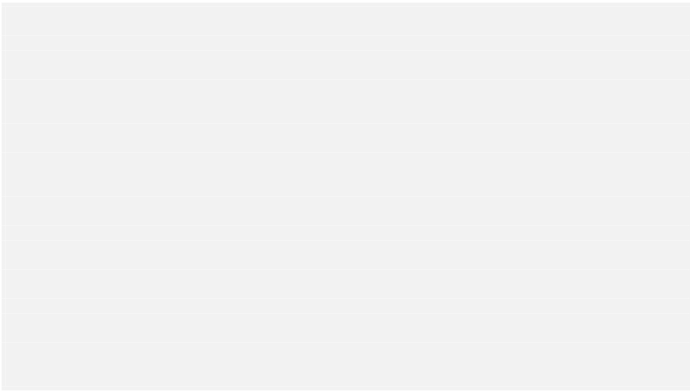
**请参考**

ST\_Node, ST\_Dump

**8.11.19**

**ST\_Node**

ST\_Node — 对LINESTRING对象添加节点进行分隔



PostGIS 2.2.0dev 开发手册

330 / 686

**用法**

geometry ST\_Node(geometry geom);

**描述**

在保持原有输入的几何对象的所有点情况下，添加最少的可能的节点到LINESTRING上

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

可用版本: 2.0.0 -需要GEOS版本>= 3.3.0.

**注意**

Due to a bug in GEOS up to 3.3.1 this function fails to node self-intersecting lines. This is ﬁxed with GEOS 3.3.2 or

higher.

**样例**

SELECT ST\_AsEWKT(

ST\_Node(’LINESTRINGZ(0 0 0, 10 10 10, 0 10 5, 10 0 3)’::geometry)

) As

output;

output

-----------

MULTILINESTRING((0 0 0,5 5 4.5),(5 5 4.5,10 10 10,0 10 5,5 5 4.5),(5 5 4.5,10 0 3))

**请参考**

ST\_UnaryUnion

**8.11.20**

**ST\_OffsetCurve**

ST\_OffsetCurve —根据一个输入的线和偏移的距离，返回一个偏移后的线。对于计算中心线的平行线很有用

**用法**

geometry ST\_OffsetCurve(geometry line, ﬂoat signed\_distance, text style\_parameters=”);

**描述**

根据一个输入的线和偏移的距离，返回一个偏移后的线。对于计算中心线的平行线很有用

如果输入的距离值是一个正值，那么会向输入线的左边偏移，如果是一个负值，那么将向反方向的右侧偏移

可用版本: 2.0 -需要GEOS版本>= 3.2, 在版本GEOS >= 3.3帮助下有了提升

第三个参数（目前只支持geometry类型）：可以指定一个分隔四分之一圆的弧段的数量（默认四分之圆做7次分隔，即8个弧段）或者空格分隔键值对参数：

’quad\_segs=#’ : 用来近似四分之一圆的弧段数，默认是8

’endcap=round|ﬂat|square’ : endcap 方式 (默认是"round", 需要 GEOS-3.2 或更高版本支持). ’butt’ 和“flat” 意义等价，也可以是参数值

’join=round|mitre|bevel’ : join 方式 (默认是"round", 需要 GEOS-3.2 或更高版本支持). ’miter’ 和’mitre’意义等价，也可以是参数值

’mitre\_limit=#.#’ : 切割率限制 (只影响miter 方式的join). ’miter\_limit’ 和’mitre\_limit’ 意义等价，也可以是参数值



PostGIS 2.2.0dev 开发手册

331 / 686

距离单位以相关的空间参考系规定的距离单位为准.

输入对象只能是 LINESTRINGS.

该函数由GEOS模块支持.

**注意**

该函数会忽略第3维坐标（Z坐标），即便输入的是一个3维对象，返回的也是2维结果

**样例**

Compute an open buffer around roads

SELECT ST\_Union(

ST\_OffsetCurve(f.the\_geom,

f.width/2, ’quad\_segs=4 join=round’),

ST\_OffsetCurve(f.the\_geom, -f.width/2, ’quad\_segs=4 join=round’)

) as track

FROM someroadstable;



PostGIS 2.2.0dev 开发手册

332 / 686

15, ’quad\_segs=4 join=round’ original line and its offset 15

units.

SELECT ST\_AsText(ST\_OffsetCurve( ←

ST\_GeomFromText(

-15, ’quad\_segs=4 join=round’ original line and its offset

-15 units

’LINESTRING(164 16,144 16,124 16,104

16,84 16,64 16,

44 16,24 16,20 16,18 16,17 17,

←

SELECT ST\_AsText(ST\_OffsetCurve(geom,

-15, ’quad\_segs=4 join=round’))

←

100,

16 18,16 20,16 40,16 60,16 80,16

16 120,16 140,16 160,16 180,16

←

←

As notsocurvy

FROM ST\_GeomFromText(

’LINESTRING(164 16,144 16,124 16,104

←

195)’),

16,84 16,64 16,

15, ’quad\_segs=4 join=round’));

--output --

44 16,24 16,20 16,18 16,17 17,

16 18,16 20,16 40,16 60,16 80,16

←

LINESTRING(164 1,18 1,12.2597485145237

2.1418070123307,

←

100,

16 120,16 140,16 160,16 180,16

←

7.39339828220179

5.39339828220179,

5.39339828220179

7.39339828220179,

2.14180701233067

←

←

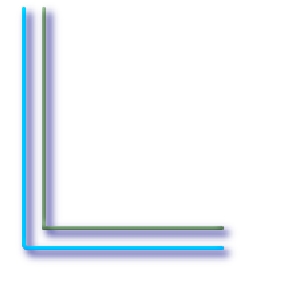
←

195)’) As geom;

-- notsocurvy --

LINESTRING(31 195,31 31,164 31)

12.2597485145237,1 18,1 195)



PostGIS 2.2.0dev 开发手册

333 / 686

double-offset to get more curvy,combined with regular

offset 15 to get parallel lines. Overlaid with original.

SELECT ST\_AsText(ST\_Collect(

ST\_OffsetCurve(geom, 15, ’ ←

double-offset to get more curvy, note the ﬁrst reverses

direction, so -30 + 15 = -15

quad\_segs=4 join=round’),

ST\_OffsetCurve(ST\_OffsetCurve( ←

geom,

SELECT ST\_AsText(ST\_OffsetCurve( ←

ST\_OffsetCurve(geom,

-30, ’quad\_segs=4 join=round’),

-15, ’quad\_segs=4 join=round’)

←

-30, ’quad\_segs=4 join=round’),

←

)

-15, ’quad\_segs=4 join=round’)) As morecurvy As parallel\_curves

FROM ST\_GeomFromText(

FROM ST\_GeomFromText(

’LINESTRING(164 16,144 16,124 16,104

16,84 16,64 16,

44 16,24 16,20 16,18 16,17 17,

←

’LINESTRING(164 16,144 16,124 16,104

16,84 16,64 16,

44 16,24 16,20 16,18 16,17 17,

←

16 18,16 20,16 40,16 60,16 80,16

100,

16 120,16 140,16 160,16 180,16

195)’) As geom;

-- morecurvy --

←

←

16 18,16 20,16 40,16 60,16 80,16

100,

16 120,16 140,16 160,16 180,16

195)’) As geom;

-- parallel curves

--

←

←

LINESTRING(164 31,46 31,40.2597485145236

←

MULTILINESTRING((164 1,18

←

32.1418070123307,

1,12.2597485145237 2.1418070123307,

35.3933982822018 35.3933982822018,

7.39339828220179

←

32.1418070123307 40.2597485145237,31

←

5.39339828220179,5.39339828220179 7.393398282201

46,31 195)

2.14180701233067 12.2597485145237,1 18,1

195),

←

-8(164 31,46 31,40.2597485145236

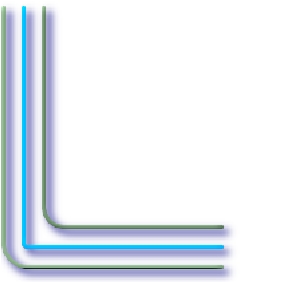
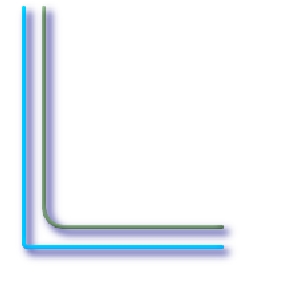
←

32.1418070123307,35.3933982822018 35.39339828220

32.1418070123307 40.2597485145237,31

46,31 195))

←



)

PostGIS 2.2.0dev 开发手册

334 / 686

15,-15 collected, join=mitre mitre\_limit=2.1

15, ’quad\_segs=4 join=bevel’ shown with original line

SELECT ST\_AsText(ST\_Collect(

SELECT ST\_AsText(ST\_OffsetCurve( ←

ST\_GeomFromText(

’LINESTRING(164 16,144 16,124 16,104

16,84 16,64 16,

44 16,24 16,20 16,18 16,17 17,

←

ST\_OffsetCurve(geom, 15, ’ ←

quad\_segs=4 join=mitre mitre\_limit=2.2’),

ST\_OffsetCurve(geom, -15, ’ ←

quad\_segs=4 join=mitre mitre\_limit=2.2’)

) )

100,

16 18,16 20,16 40,16 60,16 80,16

←

FROM ST\_GeomFromText(

’LINESTRING(164 16,144 16,124 16,104

←

195)’),

16 120,16 140,16 160,16 180,16 ←

15, ’quad\_segs=4 join= ←

16,84 16,64 16,

44 16,24 16,20 16,18 16,17 17,

16 18,16 20,16 40,16 60,16 80,16

←

bevel’));

-- output --

100,

16 120,16 140,16 160,16 180,16

←

LINESTRING(164 1,18 1,7.39339828220179

5.39339828220179,

5.39339828220179 ←

7.39339828220179,1 18,1 195)

←

195)’) As geom;

-- output --

MULTILINESTRING((164 1,11.7867965644036

1,1 11.7867965644036,1 195),

←

(31 195,31 31,164 31))

**请参考**

ST\_Buffer

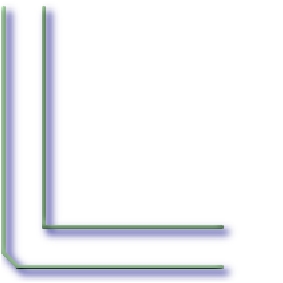
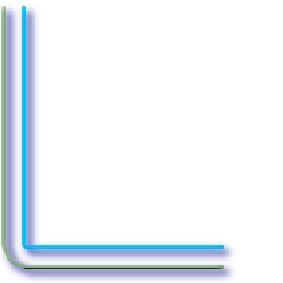
**8.11.21**

**ST\_RemoveRepeatedPoints**

ST\_RemoveRepeatedPoints —返回一个删除重复点的几何对象

**用法**

geometry ST\_RemoveRepeatedPoints(geometry geom);



PostGIS 2.2.0dev 开发手册

335 / 686

**描述**

返回一个删除重复点的几何对象。该函数实际上只支持 (multi)lines、(multi)polygons 和 multipoints类型对象，但是你可以安全地对任何几何类型对象使用该函数。因为该函数会输入的geometry collection对象做一层又一层的类型转换，直到符合要求

可用版本: 2.0.0

该函数支持 Polyhedral Surface类型几何对象.

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

**请参考**

ST\_Simplify

**8.11.22**

**ST\_SharedPaths**

ST\_SharedPaths —返回两个输入的linestrings/multilinestrings类型对象公用的path路径代表的collection对象

**用法**

geometry ST\_SharedPaths(geometry lineal1, geometry lineal2);

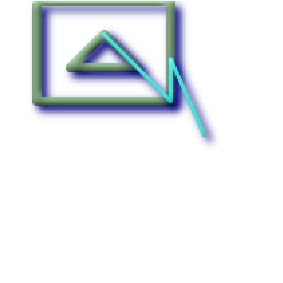
**描述**

返回两个输入的linestrings/multilinestrings类型对象公用的path路径代表的collection对象。方向一致的几何对象放在第一个集合元素内，方向相反的放在第二个集合内。公用的path和第一个几何输入几何对象一致

可用版本: 2.0.0需要GEOS版本>= 3.3.0.

**样例: Finding shared paths**

A multilinestring and a linestring



PostGIS 2.2.0dev 开发手册

336 / 686

The shared path of multilinestring and linestring overlaid with original geometries.

SELECT ST\_AsText(

ST\_SharedPaths(

ST\_GeomFromText(’MULTILINESTRING((26 125,26 200,126 200,126 125,26 125),

(51 150,101 150,76 175,51 150))’),

ST\_GeomFromText(’LINESTRING(151 100,126 156.25,126 125,90 161, 76 175)’)

)

) As wkt

wkt

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

GEOMETRYCOLLECTION(MULTILINESTRING((126 156.25,126 125),

(101 150,90 161),(90 161,76 175)),MULTILINESTRING EMPTY)

-- same example but linestring orientation flipped

SELECT ST\_AsText(

ST\_SharedPaths(

ST\_GeomFromText(’LINESTRING(76 175,90 161,126 125,126 156.25,151 100)’),

ST\_GeomFromText(’MULTILINESTRING((26 125,26 200,126 200,126 125,26 125),

(51 150,101 150,76 175,51 150))’)

)

) As wkt

wkt

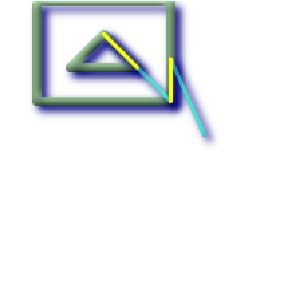
-------------------------------------------------------------

GEOMETRYCOLLECTION(MULTILINESTRING EMPTY,

MULTILINESTRING((76 175,90 161),(90 161,101 150),(126 125,126 156.25)))

**请参考**

ST\_Dump, ST\_GeometryN, ST\_NumGeometries



PostGIS 2.2.0dev 开发手册

337 / 686

**8.11.23**

**ST\_Shift\_Longitude**

ST\_Shift\_Longitude — 读取每个几何对象的点或顶点的坐标，如果经度小于0，则加上360，如果大于0则不加。该函数返回的经纬度结果在0到360度范围内以便在平面地图内展现

**用法**

geometry ST\_Shift\_Longitude(geometry geomA);

**描述**

读取每个几何对象的点或顶点的坐标，如果经度小于0，则加上360，如果大于0则不加。该函数返回的经纬度结果在0到360度范围内以便在平面地图内展现

**注意**

该函数只适用于经纬度坐标，比如WGS84参考系的经纬度坐标。

1.3.4版本之前，该函数不支持MULTIPOINT类型，1.3.4+ 版本支持MULTIPOINT类型。

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

版本提升: 2.0.0 该函数引入对Polyhedral Surface 和 TIN类型对象的支持

该函数支持 Polyhedral Surface类型几何对象.

该函数支持 Triangles 和 Triangulated Irregular Network Surfaces (TIN)类型.

**样例**

--3d points

SELECT ST\_AsEWKT(ST\_Shift\_Longitude(ST\_GeomFromEWKT(’SRID=4326;POINT(-118.58 38.38 10)’)))

As geomA,

←

ST\_AsEWKT(ST\_Shift\_Longitude(ST\_GeomFromEWKT(’SRID=4326;POINT(241.42 38.38 10)’))) As

geomb

←

geomA

geomB

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

SRID=4326;POINT(241.42 38.38 10) SRID=4326;POINT(-118.58 38.38 10)

--regular line string

SELECT ST\_AsText(ST\_Shift\_Longitude(ST\_GeomFromText(’LINESTRING(-118.58 38.38, -118.20

38.45)’)))

st\_astext

----------

LINESTRING(241.42 38.38,241.8 38.45)

**请参考**

ST\_GeomFromEWKT, ST\_GeomFromText, ST\_AsEWKT

←



PostGIS 2.2.0dev 开发手册

338 / 686

**8.11.24**

**ST\_Simplify**

ST\_Simplify — 使用Douglas-Peucker 算法简化一个给定的几何对象，并返回

**用法**

geometry ST\_Simplify(geometry geomA, ﬂoat tolerance);

**描述**

使用Douglas-Peucker 算法简化一个给定的几何对象，并返回。该函数实际上只支持 (multi)lines、(multi)polygons 和 multipoints类型对象，但是你可以安全地对任何几何类型对象使用该函数。因为该函数会输入的geometry collection对象做一层又一层的类型转换，直到符合要求

**注意**

返回的几何对象可能会丢失简单性(参考 ST\_IsSimple)

**注意**

注意输入对象拓扑结构可能会变化，也可能会产生无效的几何类型对象，使用函数(参考 ST\_SimplifyPreserveTopology) 来维持输入对象的拓扑结构。

该函数由GEOS模块支持.

可用版本: 1.2.2

**样例**

A circle simpliﬁed too much becomes a triangle, medium an octagon,

SELECT ST\_Npoints(the\_geom) As np\_before, ST\_NPoints(ST\_Simplify(the\_geom,0.1)) As

←

np01\_notbadcircle, ST\_NPoints(ST\_Simplify(the\_geom,0.5)) As np05\_notquitecircle,

ST\_NPoints(ST\_Simplify(the\_geom,1)) As np1\_octagon, ST\_NPoints(ST\_Simplify(the\_geom,10)) As ←

np10\_triangle,

(ST\_Simplify(the\_geom,100) is null) As

np100\_geometrygoesaway

FROM (SELECT ST\_Buffer(’POINT(1 3)’, 10,12) As the\_geom) As foo;

-result

np\_before | np01\_notbadcircle | np05\_notquitecircle | np1\_octagon | np10\_triangle |

np100\_geometrygoesaway

←

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

49 |

33 |

17 |

9 |

4 | t

**请参考**

ST\_IsSimple, ST\_SimplifyPreserveTopology, Topology ST\_Simplify

**8.11.25**

**ST\_SimplifyPreserveTopology**

ST\_SimplifyPreserveTopology —使用Douglas-Peucker算法，根据给定的几何对象，返回一个简化版的几何对象。该函数会避免创建新的派生的无效的几何对象（特别是多边形）



PostGIS 2.2.0dev 开发手册

339 / 686

**用法**

geometry ST\_SimplifyPreserveTopology(geometry geomA, ﬂoat tolerance);

**描述**

使用Douglas-Peucker算法，根据给定的几何对象，返回一个简化版的几何对象。该函数会避免创建新的派生的无效的几何对象（特别是多边形）。该函数实际上只支持 (multi)lines、(multi)polygons 类型对象，但是你可以安全地对任何几何类型对象使用该函数。因为该函数会输入的geometry collection对象做一层又一层的类型转换，直到符合要求

该函数由GEOS模块支持.

**注意**

需要 GEOS 3.0.0+版本支持

可用版本: 1.3.3

**样例**

Same example as Simplify, but we see Preserve Topology prevents oversimpliﬁcation. The circle can at most become a square.

SELECT ST\_Npoints(the\_geom) As np\_before, ST\_NPoints(ST\_SimplifyPreserveTopology(the\_geom ←

,0.1)) As np01\_notbadcircle, ST\_NPoints(ST\_SimplifyPreserveTopology(the\_geom,0.5)) As

np05\_notquitecircle,

ST\_NPoints(ST\_SimplifyPreserveTopology(the\_geom,1)) As np1\_octagon, ST\_NPoints( ←

ST\_SimplifyPreserveTopology(the\_geom,10)) As np10\_square,

←

ST\_NPoints(ST\_SimplifyPreserveTopology(the\_geom,100)) As

np100\_stillsquare

FROM (SELECT ST\_Buffer(’POINT(1 3)’, 10,12) As the\_geom) As foo;

--result--

np\_before | np01\_notbadcircle | np05\_notquitecircle | np1\_octagon | np10\_square |

np100\_stillsquare

←

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

49 |

33 |

17 |

9 |

5 |

←

5

**请参考**

ST\_Simplify

**8.11.26**

**ST\_Split**

ST\_Split — 通过切割一个几何对象，返回一个collection 几何对象。

**用法**

geometry ST\_Split(geometry input, geometry blade);



PostGIS 2.2.0dev 开发手册

340 / 686

**描述**

该函数会切割一个几何对象，用点切割线，用线切割线，用线切割多边形。返回一个collection类型对象。可以把该函数当做函数ST\_Union的相反函数。理论上用ST\_Union函数到函数ST\_Split 返回的collection上面始终能够得到原始的几何对象

可用版本: 2.0.0

**注意**

为了提高函数ST\_Split的鲁棒性，可以使用以一个非常低的误差用blade参数代表的几何对象来切割第一个参数代表的几何对象。否则该函数使用内部网格时候可能会引起一些误差问题，即切割几何对象的坐标和被切割的机会对象坐标不能落在一起，导致输入对象不能被正确地切割。(参考 #2192).

**样例**

Polygon Cut by Line

Before Split

After split

-- this creates a geometry collection consisting of the 2 halves of the polygon

-- this is similar to the example we demonstrated in ST\_BuildArea

SELECT ST\_Split(circle, line)

FROM (SELECT

ST\_MakeLine(ST\_MakePoint(10, 10),ST\_MakePoint(190, 190)) As line,

ST\_Buffer(ST\_GeomFromText(’POINT(100 90)’), 50) As circle) As foo;

-- result --

GEOMETRYCOLLECTION(POLYGON((150 90,149.039264020162 80.2454838991936,146.193976625564

70.8658283817455,..), POLYGON(..)))

-- To convert to individual polygons, you can use ST\_Dump or ST\_GeometryN

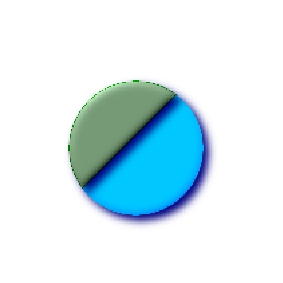
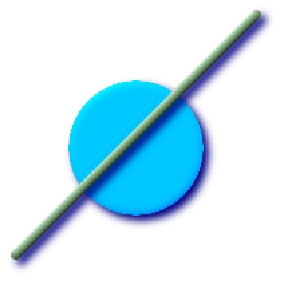
SELECT ST\_AsText((ST\_Dump(ST\_Split(circle, line))).geom) As wkt

FROM (SELECT

ST\_MakeLine(ST\_MakePoint(10, 10),ST\_MakePoint(190, 190)) As line,

ST\_Buffer(ST\_GeomFromText(’POINT(100 90)’), 50) As circle) As foo;

←



PostGIS 2.2.0dev 开发手册

341 / 686

-- result --

wkt

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

POLYGON((150 90,149.039264020162 80.2454838991936,..))

POLYGON((60.1371179574584 60.1371179574584,58.4265193848728

62.2214883490198,53.8060233744357 ..))

Multilinestring Cut by point

Before Split

←

After split

SELECT ST\_AsText(ST\_Split(mline, pt)) As wktcut

FROM (SELECT

ST\_GeomFromText(’MULTILINESTRING((10 10, 190 190), (15 15, 30 30, 100 90))’) As mline,

ST\_Point(30,30) As pt) As foo;

wktcut

------

GEOMETRYCOLLECTION(

LINESTRING(10 10,30 30),

LINESTRING(30 30,190 190),

LINESTRING(15 15,30 30),

LINESTRING(30 30,100 90)

)

**请参考**

ST\_AsText, ST\_BuildArea, ST\_Dump, ST\_GeometryN, ST\_Union

**8.11.27**

**ST\_SymDifference**

ST\_SymDifference —返回输入对象A和B union后再去除A和B相交的部分。该函数是一个对称性函数，因为下面等式成立

ST\_SymDifference(A,B) = ST\_SymDifference(B,A).



PostGIS 2.2.0dev 开发手册

342 / 686

**用法**

geometry ST\_SymDifference(geometry geomA, geometry geomB);

**描述**

返回输入对象A和B union后再去除A和B相交的部分。该函数是一个对称性函数，因为下面等式成立

ST\_SymDifference(A,B) = ST\_SymDifference(B,A)。你可以认为该函数的返回值等于 ST\_Union(geomA,geomB) - ST\_Intersection(A,B).

该函数由GEOS模块支持

**注意**

该函数不支持GeometryCollection参数

该函数方法实现了规范 OpenGIS Simple Features Implementation Speciﬁcation for SQL 1.1. s2.1.1.3

该函数方法实现了规范 SQL/MM speciﬁcation. SQL-MM 3: 5.1.21

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）。然而该函数在处理时候只考虑x,y坐标，并返回Z坐标

**样例**

The original linestrings shown together

--Safe for 2d - symmetric difference of 2 linestrings

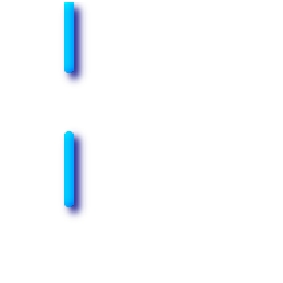
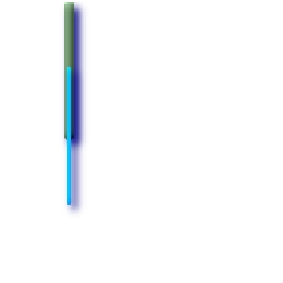
SELECT ST\_AsText(

ST\_SymDifference(

ST\_GeomFromText(’LINESTRING(50 100, 50 200)’),

ST\_GeomFromText(’LINESTRING(50 50, 50 150)’)

The symmetric difference of the two linestrings



PostGIS 2.2.0dev 开发手册

343 / 686

)

);

st\_astext

---------

MULTILINESTRING((50 150,50 200),(50 50,50 100))

--When used in 3d doesn’t quite do the right thing

SELECT ST\_AsEWKT(ST\_SymDifference(ST\_GeomFromEWKT(’LINESTRING(1 2 1, 1 4 2)’),

ST\_GeomFromEWKT(’LINESTRING(1 1 3, 1 3 4)’)))

st\_astext

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

MULTILINESTRING((1 3 2.75,1 4 2),(1 1 3,1 2 2.25))

**请参考**

ST\_Difference, ST\_Intersection, ST\_Union

**8.11.28**

**ST\_Union**

ST\_Union — 返回一个代表一些几何对象的并集的几何对象

**用法**

geometry ST\_Union(geometry set g1ﬁeld);

geometry ST\_Union(geometry g1, geometry g2);

geometry ST\_Union(geometry[] g1\_array);

**描述**

该函数输出对象可以是MULTI\*,单例几何对象或geometry collection类型对象。该函数有两种形式。形式1返回两个输入几何对象的并集对应的几何对象。形式2是一个聚合函数，它根据一个几何对象的集合，对他们求并集

聚合版本: 该函数根据输入的几何对象集合返回一个MULTI 几何对象或非 MULTI 几何对象。ST\_Union()

该函数是聚合函数意味着，对于多行的数据，该函数的作用方式会像SQL里面的聚合函数SUM() and AVG() 一样，该函数会忽略掉NULL几何类型对象

非聚合版本：该函数会返回两个输入对象的并集几何对象。输出类型可以是MULTI\*,NON-MULTI 或 GEOMETRYCOLLECTION。如果任意一个输入对象是NULL，返回值也是NULL

**注意**

函数ST\_Collect 和函数 ST\_Union 经常可以互换使用。函数ST\_Collect一般来说要比函数ST\_Union 快很多，因为ST\_Collect函数不会去分解输入几何对象的边界或者检查一个MultiPolgon对象是否有重叠部分。

该函数由GEOS模块支持.

注意:该函数原先叫做GeomUnion(),而函数GeomUnion()原先的名字是“UNION”，由于UNION是SQL的保留字，所以就各种改啦



PostGIS 2.2.0dev 开发手册

344 / 686

可用版本: 1.4.0 - ST\_Union性能有所提升。引入了ST\_Union(geomarray) 支持，在PostgreSQL中运行得也更快。如果你使用

GEOS 3.1.0+版本，函数 ST\_Union 会使用更快的Cascaded Union 算法，该算法介绍如下http://blog.cleverelephant.ca/

[2009/01/must-faster-unions-in-postgis-14.html](http://blog.cleverelephant.ca/2009/01/must-faster-unions-in-postgis-14.html)

该函数方法实现了规范 OpenGIS Simple Features Implementation Speciﬁcation for SQL 1.1. s2.1.1.3

**注意**

聚合版本没有在OGC SPEC做明确的定义

该函数方法实现了规范 SQL/MM speciﬁcation. SQL-MM 3: 5.1.19 the z-index (elevation) when polygons are

involved.

**样例**

Aggregate example

SELECT stusps,

ST\_Multi(ST\_Union(f.the\_geom)) as singlegeom

FROM sometable As f

GROUP BY stusps

Non-Aggregate example

SELECT ST\_AsText(ST\_Union(ST\_GeomFromText(’POINT(1 2)’),

ST\_GeomFromText(’POINT(-2 3)’) ) )

st\_astext

----------

MULTIPOINT(-2 3,1 2)

SELECT ST\_AsText(ST\_Union(ST\_GeomFromText(’POINT(1 2)’),

ST\_GeomFromText(’POINT(1 2)’) ) );

st\_astext

----------

POINT(1 2)

--3d example - sort of supports 3d (and with mixed dimensions!)

SELECT ST\_AsEWKT(st\_union(the\_geom))

FROM

(SELECT ST\_GeomFromEWKT(’POLYGON((-7 4.2,-7.1 4.2,-7.1 4.3,

-7 4.2))’) as the\_geom

UNION ALL

SELECT ST\_GeomFromEWKT(’POINT(5 5 5)’) as the\_geom

UNION ALL

SELECT ST\_GeomFromEWKT(’POINT(-2 3 1)’) as the\_geom

UNION ALL

SELECT ST\_GeomFromEWKT(’LINESTRING(5 5 5, 10 10 10)’) as the\_geom ) as foo;

st\_asewkt

---------

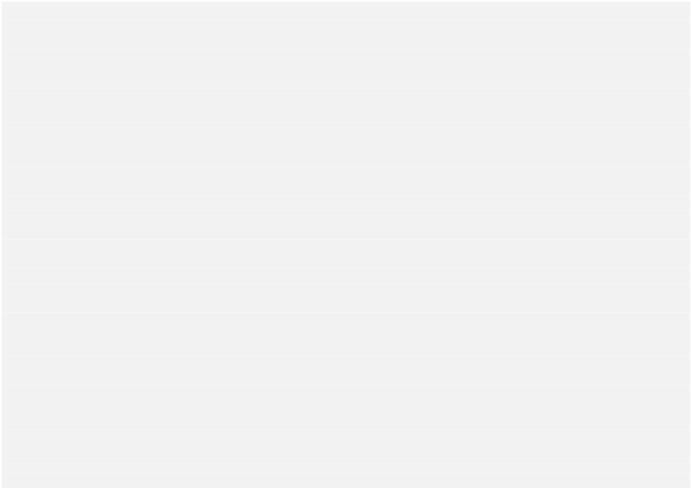
GEOMETRYCOLLECTION(POINT(-2 3 1),LINESTRING(5 5 5,10 10 10),POLYGON((-7 4.2 5,-7.1 4.2

5,-7.1 4.3 5,-7 4.2 5)));

--3d example not mixing dimensions

SELECT ST\_AsEWKT(st\_union(the\_geom))

←



PostGIS 2.2.0dev 开发手册

345 / 686

FROM

(SELECT ST\_GeomFromEWKT(’POLYGON((-7 4.2 2,-7.1 4.2 3,-7.1 4.3 2,

-7 4.2 2))’) as the\_geom

UNION ALL

SELECT ST\_GeomFromEWKT(’POINT(5 5 5)’) as the\_geom

UNION ALL

SELECT ST\_GeomFromEWKT(’POINT(-2 3 1)’) as the\_geom

UNION ALL

SELECT ST\_GeomFromEWKT(’LINESTRING(5 5 5, 10 10 10)’) as the\_geom ) as foo;

st\_asewkt

---------

GEOMETRYCOLLECTION(POINT(-2 3 1),LINESTRING(5 5 5,10 10 10),POLYGON((-7 4.2 2,-7.1 4.2

3,-7.1 4.3 2,-7 4.2 2)))

--样例 using new Array construct

SELECT ST\_Union(ARRAY(SELECT the\_geom FROM sometable));

SELECT ST\_AsText(ST\_Union(ARRAY[ST\_GeomFromText(’LINESTRING(1 2, 3 4)’),

ST\_GeomFromText(’LINESTRING(3 4, 4 5)’)])) As wktunion;

--wktunion---

MULTILINESTRING((3 4,4 5),(1 2,3 4))

**请参考**

ST\_Collect ST\_UnaryUnion

←

**8.11.29**

**ST\_UnaryUnion**

ST\_UnaryUnion — 与函数ST\_Union类似，但处理geometry类型的对象

**用法**

geometry ST\_UnaryUnion(geometry geom);

**描述**

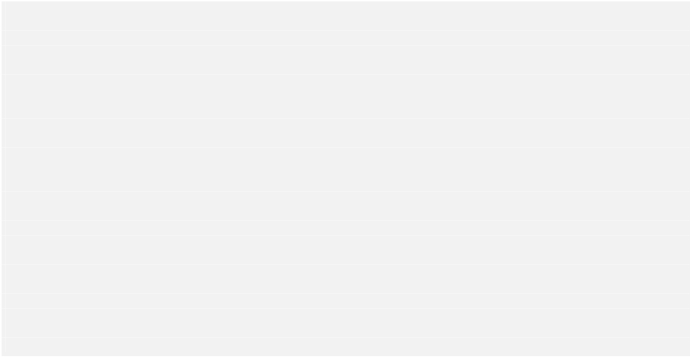
与函数ST\_Union不同的是，函数ST\_UnaryUnion会有效输入的MULTIPOLYGON对象公共的边界做分解。然后对他们求并集。每一个输入几何对象被假定是有效的。你可以用这个函数来对一些LINESTRING对象添加节点（分割点）。你可能会混合使用ST\_UnaryUnion 和 ST\_Collect来确定你要在有限的内存和CPU中一次最佳分解多少几何对象，找到在ST\_Union与ST\_MemUnion中一种平衡

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

可用版本: 2.0.0 -需要GEOS版本>= 3.3.0.

**请参考**

ST\_Union ST\_MemUnion ST\_Collect



PostGIS 2.2.0dev 开发手册

346 / 686

**8.12**

**8.12.1**

**线性函数参考**

**ST\_LineInterpolatePoint**

ST\_LineInterpolatePoint — 返回在一个LINESTRING对象上面在指定位置上插入（获取）的一个点的几何对象。第二个参数范围是0到1内的float8类型，表示被插入的点在整个LINESTRING的位置。（译者注：比如0.5是在中心位置插入，0.333可以认为是在1/3位置插入）

**用法**

geometry ST\_LineInterpolatePoint(geometry a\_linestring, ﬂoat a\_fraction);

**描述**

返回在一个LINESTRING对象上面在指定位置上插入（获取）的一个点的几何对象。第二个参数范围是0到1内的float8类型，表示被插入的点在整个LINESTRING的位置。（译者注：比如0.5是在中心位置插入，0.333可以认为是在1/3位置插入）

参考 ST\_LineLocatePoint 计算离LINESTRING最近的Point对象

**注意**

从1.1.1版本开始，该函数也可以处理带M和Z值的坐标，而之前的版本只把这两个坐标值都设置为0.0

可用版本: 0.8.2, 在1.1.1版本添加了对Z 和 M 坐标支持

版本变更: 2.1.0. 在2.0.x版本中，该函数被称为ST\_LineInterpolatePoint.

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

**样例**

A linestring with the interpolated point at 20% position (0.20)



PostGIS 2.2.0dev 开发手册

347 / 686

--Return point 20% along 2d line

SELECT ST\_AsEWKT(ST\_LineInterpolatePoint(the\_line, 0.20))

FROM (SELECT ST\_GeomFromEWKT(’LINESTRING(25 50, 100 125, 150 190)’) as the\_line) As foo;

st\_asewkt

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

POINT(51.5974135047432 76.5974135047432)

--Return point mid-way of 3d line

SELECT ST\_AsEWKT(ST\_LineInterpolatePoint(the\_line, 0.5))

FROM (SELECT ST\_GeomFromEWKT(’LINESTRING(1 2 3, 4 5 6, 6 7 8)’) as the\_line) As foo;

st\_asewkt

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

POINT(3.5 4.5 5.5)

--find closest point on a line to a point or other geometry

SELECT ST\_AsText(ST\_LineInterpolatePoint(foo.the\_line, ST\_LineLocatePoint(foo.the\_line,

ST\_GeomFromText(’POINT(4 3)’))))

FROM (SELECT ST\_GeomFromText(’LINESTRING(1 2, 4 5, 6 7)’) As the\_line) As foo;

st\_astext

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

POINT(3 4)

**请参考**

ST\_AsText, ST\_AsEWKT, ST\_Length, ST\_LineLocatePoint

←

**8.12.2**

**ST\_LineLocatePoint**

ST\_LineLocatePoint —根据输入的点和LINESTRING，返回这个点在LINESTRING上的位置（如果点不在这个LINESTRING上，就返回最近的）

**用法**

ﬂoat ST\_LineLocatePoint(geometry a\_linestring, geometry a\_point);

**描述**

根据输入的点和LINESTRING，返回这个点在LINESTRING上的位置（如果点不在这个LINESTRING上，就返回最近的）

你可以用得到的位置，反过来用函数 (ST\_LineInterpolatePoint) 来获得插值点，或者函数 (ST\_LineSubstring)来截取一部分LINESTRING.

可用版本: 1.1.0

版本变更: 2.1.0. Up to 2.0.x this was called ST\_Line\_Locate\_Point.



PostGIS 2.2.0dev 开发手册

348 / 686

**样例**

--Rough approximation of finding the street number of a point along the street

--注意 the whole foo thing is just to generate dummy data that looks

--like house centroids and street

--We use ST\_DWithin to exclude

--houses too far away from the street to be considered on the street

SELECT ST\_AsText(house\_loc) As as\_text\_house\_loc,

startstreet\_num +

CAST( (endstreet\_num - startstreet\_num)

\* ST\_LineLocatePoint(street\_line, house\_loc) As integer) As street\_num

FROM

(SELECT ST\_GeomFromText(’LINESTRING(1 2, 3 4)’) As street\_line,

ST\_MakePoint(x\*1.01,y\*1.03) As house\_loc, 10 As startstreet\_num,

20 As endstreet\_num

FROM generate\_series(1,3) x CROSS JOIN generate\_series(2,4) As y)

As foo

WHERE ST\_DWithin(street\_line, house\_loc, 0.2);

as\_text\_house\_loc | street\_num

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

POINT(1.01 2.06) | 10

POINT(2.02 3.09) | 15

POINT(3.03 4.12) | 20

--find closest point on a line to a point or other geometry

SELECT ST\_AsText(ST\_LineInterpolatePoint(foo.the\_line, ST\_LineLocatePoint(foo.the\_line,

ST\_GeomFromText(’POINT(4 3)’))))

FROM (SELECT ST\_GeomFromText(’LINESTRING(1 2, 4 5, 6 7)’) As the\_line) As foo;

st\_astext

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

POINT(3 4)

**请参考**

ST\_DWithin, ST\_Length2D, ST\_LineInterpolatePoint, ST\_LineSubstring

←

**8.12.3**

**ST\_LineSubstring**

ST\_LineSubstring — 从一个LINESTRING钟返回一个从输入的LINESTRING提取指定起点和终点的片段，第二个和第三个参数都是float8类型，值范围在0到1。译者注：该函数其实就是切割一个LINESTRING对象片段，像提取字符串一样。

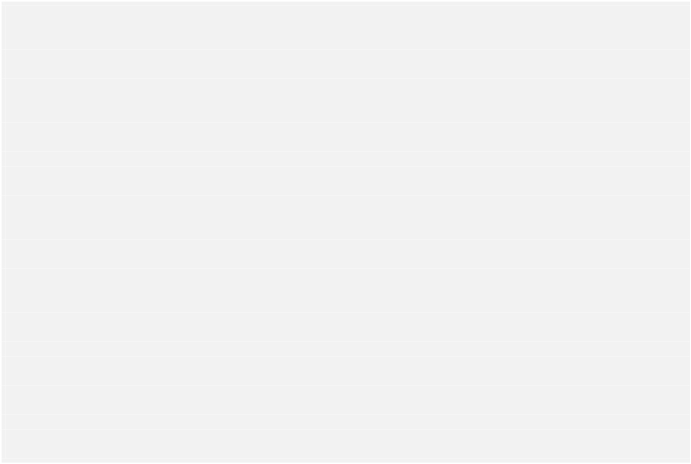
**用法**

geometry ST\_LineSubstring(geometry a\_linestring, ﬂoat startfraction, ﬂoat endfraction);

**描述**

从一个LINESTRING钟返回一个从输入的LINESTRING提取指定起点和终点的片段，第二个和第三个参数都是float8类型，值范围在0到1。（译者注：该函数其实就是切割一个LINESTRING对象片段，像提取字符串一样。）该函数只适用于LINESTRING类型对象，如果想对连续的MULTILINESTRING类型对象使用，需要与函数ST\_LineMerge一起使用。如果参数的起始点和终点一样，那么返回值和ST\_LineInterpolatePoint返回值一样。

参考 ST\_LineLocatePoint 返回距离一个点最近的位置



PostGIS 2.2.0dev 开发手册

349 / 686

**注意**

从1.1.1版本开始，该函数会保留M和Z的值，而先前的版本会把这两个坐标设置成未指定的值

可用版本: 1.1.0, Z and M supported added in 1.1.1

版本变更: 2.1.0. Up to 2.0.x this was called ST\_LineSubstring.

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

**样例**

A linestring seen with 1/3 midrange overlaid (0.333, 0.666)

--Return the approximate 1/3 mid-range part of a linestring

SELECT ST\_AsText(ST\_Line\_SubString(ST\_GeomFromText(’LINESTRING(25 50, 100 125, 150 190)’),

←

0.333, 0.666));

st\_astext

----------------------------------------------------------------------------------------------- - ←

LINESTRING(69.2846934853974 94.2846934853974,100 125,111.700356260683 140.210463138888)

--The below example simulates a while loop in

--SQL using PostgreSQL generate\_series() to cut all

--linestrings in a table to 100 unit segments

-- of which no segment is longer than 100 units

-- units are measured in the SRID units of measurement

-- It also assumes all geometries are LINESTRING or contiguous MULTILINESTRING

--and no geometry is longer than 100 units\*10000

--for better performance you can reduce the 10000

--to match max number of segments you expect

SELECT

CASE

WHEN

ELSE

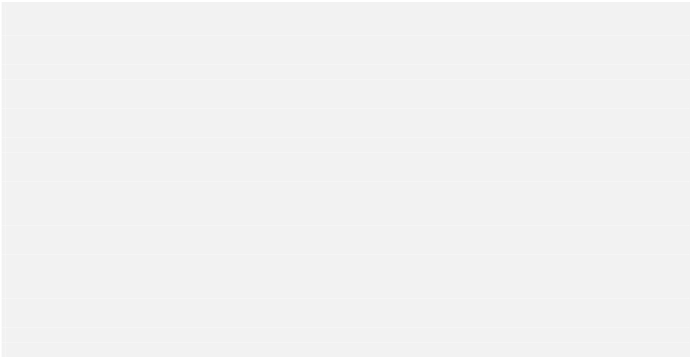
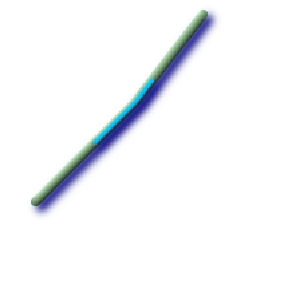
END)

field1, field2, ST\_LineSubstring(the\_geom, 100.00\*n/length,

100.00\*(n+1) < length THEN 100.00\*(n+1)/length

1

As the\_geom



PostGIS 2.2.0dev 开发手册

350 / 686

FROM

(SELECT sometable.field1, sometable.field2,

ST\_LineMerge(sometable.the\_geom) AS the\_geom,

ST\_Length(sometable.the\_geom) As length

FROM sometable

) AS t

CROSS JOIN generate\_series(0,10000) AS n

WHERE n\*100.00/length < 1;

**请参考**

ST\_Length, ST\_LineInterpolatePoint, ST\_LineMerge

**8.12.4**

**ST\_LocateAlong**

ST\_LocateAlong — 返回几何对象的M值与给定的M值一样的几何对象，目前不支持多边形类的几何对象

**用法**

geometry ST\_LocateAlong(geometry ageom\_with\_measure, ﬂoat a\_measure, ﬂoat offset);

**描述**

返回几何对象的M值与给定的M值一样的几何对象，目前不支持多边形类的几何对象

如果已经提供了偏移单位，根据提供的偏移单位，在输入的LINESTRING上向左或向右偏移。偏移值为正，则向左偏移，偏移值为负，则向右偏移。（译者注：该偏移值的作用，就是在匹配参数a\_measure值的几何上再进行偏移）

语法定义规范如下: ISO/IEC CD 13249-3:200x(E) - Text for Continuation CD Editing Meeting

可用版本: 1.1.0 by old name ST\_Locate\_Along\_Measure.

版本变更: 2.0.0 ，该函数之前的版本叫做ST\_Locate\_Along\_Measure，老的函数名不推荐使用，将来也不会在支持

**注意**

该函数只适用于带M坐标的几何对象

该函数支持M坐标.

**样例**

SELECT ST\_AsText(the\_geom)

FROM

(SELECT ST\_LocateAlong(

ST\_GeomFromText(’MULTILINESTRINGM((1 2 3, 3 4 2, 9 4 3),

(1 2 3, 5 4 5))’),3) As the\_geom) As foo;

st\_asewkt

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



PostGIS 2.2.0dev 开发手册

351 / 686

MULTIPOINT M (1 2 3)

--Geometry collections are difficult animals so dump them

--to make them more digestable

SELECT ST\_AsText((ST\_Dump(the\_geom)).geom)

FROM

(SELECT ST\_LocateAlong(

ST\_GeomFromText(’MULTILINESTRINGM((1 2 3, 3 4 2, 9 4 3),

(1 2 3, 5 4 5))’),3) As the\_geom) As foo;

st\_asewkt

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

POINTM(1 2 3)

POINTM(9 4 3)

POINTM(1 2 3)

**请参考**

ST\_Dump, ST\_LocateBetween

**8.12.5**

**ST\_LocateBetween**

ST\_LocateBetween — 根据指定的测量值范围，返回一个在输入几何对象的M坐标能匹配的的geometry collection 对象，不支持Polygon对象

**用法**

geometry ST\_LocateBetween(geometry geomA, ﬂoat measure\_start, ﬂoat measure\_end, ﬂoat offset);

**描述**

根据指定的测量值范围，返回一个在输入几何对象的M坐标能匹配的的geometry collection 对象，不支持Polygon对象

语义定义如下: ISO/IEC CD 13249-3:200x(E) - Text for Continuation CD Editing Meeting

可用版本: 1.1.0 旧的函数名ST\_Locate\_Between\_Measures.

版本变更: 2.0.0 – 在之前的版本，该函数被称作ST\_Locate\_Between\_Measures，旧的函数名不推荐使用，将来也会移除，但当前为了保持后向兼容，依然可用

该函数支持M坐标.

**样例**

SELECT ST\_AsText(the\_geom)

FROM

(SELECT ST\_LocateBetween(

ST\_GeomFromText(’MULTILINESTRING M ((1 2 3, 3 4 2, 9 4 3),

(1 2 3, 5 4 5))’),1.5, 3) As the\_geom) As foo;

st\_asewkt

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

GEOMETRYCOLLECTION M (LINESTRING M (1 2 3,3 4 2,9 4 3),POINT M (1 2 3))

--Geometry collections are difficult animals so dump them



PostGIS 2.2.0dev 开发手册

352 / 686

--to make them more digestable

SELECT ST\_AsText((ST\_Dump(the\_geom)).geom)

FROM

(SELECT ST\_LocateBetween(

ST\_GeomFromText(’MULTILINESTRING M ((1 2 3, 3 4 2, 9 4 3),

(1 2 3, 5 4 5))’),1.5, 3) As the\_geom) As foo;

st\_asewkt

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

LINESTRING M (1 2 3,3 4 2,9 4 3)

POINT M (1 2 3)

**请参考**

ST\_Dump, ST\_LocateAlong

**8.12.6**

**ST\_LocateBetweenElevations**

ST\_LocateBetweenElevations — 返回高度值在指定范围内的几何对象（collection）。目前只支持 3D, 4D LINESTRINGS 和 MULTILINESTRINGS

**用法**

geometry ST\_LocateBetweenElevations(geometry geom\_mline, ﬂoat elevation\_start, ﬂoat elevation\_end);

**描述**

返回高度值在指定范围内的几何对象（collection）。目前只支持 3D, 4D LINESTRINGS 和 MULTILINESTRINGS

可用版本: 1.4.0

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

**样例**

SELECT ST\_AsEWKT(ST\_LocateBetweenElevations(

ST\_GeomFromEWKT(’LINESTRING(1 2 3, 4 5 6)’),2,4)) As ewelev;

ewelev

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

MULTILINESTRING((1 2 3,2 3 4))

SELECT ST\_AsEWKT(ST\_LocateBetweenElevations(

ST\_GeomFromEWKT(’LINESTRING(1 2 6, 4 5 -1, 7 8 9)’),6,9)) As ewelev;

ewelev

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

GEOMETRYCOLLECTION(POINT(1 2 6),LINESTRING(6.1 7.1 6,7 8 9))

--Geometry collections are difficult animals so dump them

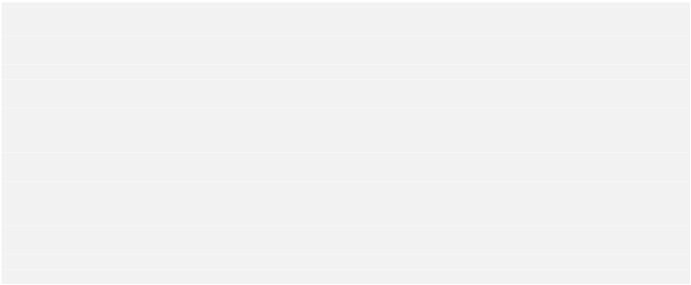
--to make them more digestable

SELECT ST\_AsEWKT((ST\_Dump(the\_geom)).geom)

FROM

(SELECT ST\_LocateBetweenElevations(

ST\_GeomFromEWKT(’LINESTRING(1 2 6, 4 5 -1, 7 8 9)’),6,9) As the\_geom) As foo;



PostGIS 2.2.0dev 开发手册

353 / 686

st\_asewkt

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

POINT(1 2 6)

LINESTRING(6.1 7.1 6,7 8 9)

**请参考**

ST\_Dump

**8.12.7**

**ST\_InterpolatePoint**

ST\_InterpolatePoint — 根据提供的点，返回输入几何对象离提供的点最近的点的M值

**用法**

ﬂoat ST\_InterpolatePoint(geometry line, geometry point);

**描述**

根据提供的点，返回输入几何对象离提供的点最近的点的M值

可用版本: 2.0.0

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

**样例**

SELECT ST\_InterpolatePoint(’LINESTRING M (0 0 0, 10 0 20)’, ’POINT(5 5)’);

st\_interpolatepoint

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

10

**请参考**

ST\_AddMeasure, ST\_LocateAlong, ST\_LocateBetween

**8.12.8**

**ST\_AddMeasure**

ST\_AddMeasure — 在输入几何对象的起点和终点之间进行线性插值，返回一个新的带有M值的几何对象，如果输入的几何对象没有测量维度的M值，那么就自动驾驶。如果输入的几何对象有M值，那么会覆盖掉。

目前该函数只支持LINESTRINGS 和 MULTILINESTRINGS

**用法**

geometry ST\_AddMeasure(geometry geom\_mline, ﬂoat measure\_start, ﬂoat measure\_end);



PostGIS 2.2.0dev 开发手册

354 / 686

**描述**

ST\_AddMeasure — 在输入几何对象的起点和终点之间进行线性插值，返回一个新的带有M值的几何对象，如果输入的几何对象没有测量维度的M值，那么就自动驾驶。如果输入的几何对象有M值，那么会覆盖掉。

目前该函数只支持LINESTRINGS 和 MULTILINESTRINGS

可用版本: 1.5.0

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

**样例**

SELECT ST\_AsText(ST\_AddMeasure(

ST\_GeomFromEWKT(’LINESTRING(1 0, 2 0, 4 0)’),1,4)) As ewelev;

ewelev

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

LINESTRINGM(1 0 1,2 0 2,4 0 4)

SELECT ST\_AsText(ST\_AddMeasure(

ST\_GeomFromEWKT(’LINESTRING(1 0 4, 2 0 4, 4 0 4)’),10,40)) As ewelev;

ewelev

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

LINESTRING(1 0 4 10,2 0 4 20,4 0 4 40)

SELECT ST\_AsText(ST\_AddMeasure(

ST\_GeomFromEWKT(’LINESTRINGM(1 0 4, 2 0 4, 4 0 4)’),10,40)) As ewelev;

ewelev

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

LINESTRINGM(1 0 10,2 0 20,4 0 40)

SELECT ST\_AsText(ST\_AddMeasure(

ST\_GeomFromEWKT(’MULTILINESTRINGM((1 0 4, 2 0 4, 4 0 4),(1 0 4, 2 0 4, 4 0 4))’),10,70)) As ←

ewelev;

ewelev

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

MULTILINESTRINGM((1 0 10,2 0 20,4 0 40),(1 0 40,2 0 50,4 0 70))

**8.13**

**长事务支持**

该章节以及相关的pl/pgsql函数用于提供 [Web Feature](http://www.opengeospatial.org/standards/wfs) [Service](http://www.opengeospatial.org/standards/wfs) 规范所要求的长事务支持实现.

**注意**

要想使用该功能，用户必须使用serializable transaction level 这样的数据库隔离级别，否则锁事务无法实现

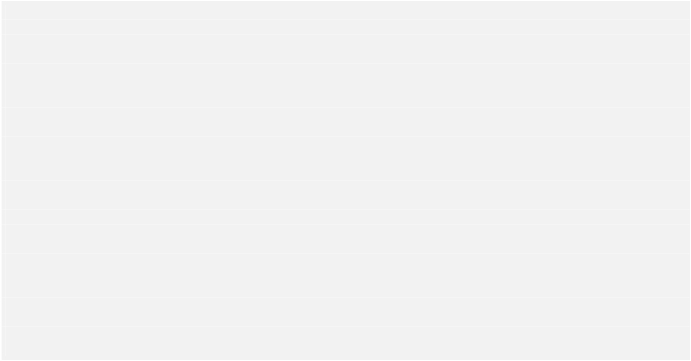
**8.13.1**

**AddAuth**

AddAuth — 添加一个用于当前事务的鉴权令牌（authorization token），如果成功返回true

**用法**

boolean AddAuth(text auth\_token);



PostGIS 2.2.0dev 开发手册

355 / 686

**描述**

添加一个用于当前事务的鉴权令牌（authorization token），如果成功返回true。

在临时表temp\_lock\_have\_table 中添加一个当前事务标识和鉴权令牌

可用版本: 1.1.3

**样例**

SELECT LockRow(’towns’, ’353’, ’priscilla’);

BEGIN TRANSACTION;

SELECT AddAuth(’joey’);

UPDATE towns SET the\_geom = ST\_Translate(the\_geom,2,2) WHERE gid = 353;

COMMIT;

---Error--

ERROR:

**请参考**

LockRow

UPDATE where "gid" = ’353’ requires authorization ’priscilla’

**8.13.2**

**CheckAuth**

CheckAuth — 基于鉴权令牌（authorization token），在一个表上创建一个阻止或允许更新或删除表记录的触发器

**用法**

integer CheckAuth(text a\_schema\_name, text a\_table\_name, text a\_key\_column\_name);

integer CheckAuth(text a\_table\_name, text a\_key\_column\_name);

**描述**

基于鉴权令牌（authorization token），在一个表上创建一个阻止或允许更新或删除表记录的触发器。通过列的<rowid\_col>方式来标识行。如果没有指定schema，使用当前current\_schema()对应的值.

**注意**

如果该表上面已经有了一个鉴权触发器，函数会返回错误。如果没有开启事务支持，函数会抛出异常

可用版本: 1.1.3

**样例**

SELECT CheckAuth(’public’, ’towns’, ’gid’);

result

------

0



PostGIS 2.2.0dev 开发手册

356 / 686

**请参考**

EnableLongTransactions

**8.13.3**

**DisableLongTransactions**

DisableLongTransactions —关闭长事务支持。该函数会移除对长事务支持所创建的元数据信息表（保存这些空间表的相关信息），并丢掉所有表上面的锁检测的触发器

**用法**

text DisableLongTransactions();

**描述**

关闭长事务支持。该函数会移除对长事务支持，并丢掉所有表上面的锁检测的触发器

具体就是该函数会删除元数据表authorization\_table 和视图authorized\_tables以及触发器checkauthtrigger

可用版本: 1.1.3

**样例**

SELECT DisableLongTransactions();

--result--

Long transactions support disabled

**请参考**

EnableLongTransactions

**8.13.4**

**EnableLongTransactions**

EnableLongTransactions —开启长事务支持。该函数会创建开启事务支持所必须的元数据表。在使用本章节其他函数（本节的函数都需要事务支持）前需要调用该函数一次，当然再执行一次也没有什么问题

**用法**

text EnableLongTransactions();

**描述**

开启长事务支持。该函数会创建开启事务支持所必须的元数据表。在使用本章节其他函数（本节的函数都需要事务支持）前需要调用该函数一次，当然再执行一次也没有什么问题

具体就是该函数会创建元数据表authorization\_table 和视图authorized\_tables

可用版本: 1.1.3

PostGIS 2.2.0dev 开发手册

357 / 686

**样例**

SELECT EnableLongTransactions();

--result--

Long transactions support enabled

**请参考**

DisableLongTransactions

**8.13.5**

**LockRow**

LockRow — 对于一个表中具体的行加锁或者说加认证

**用法**

integer LockRow(text a\_schema\_name, text a\_table\_name, text a\_row\_key, text an\_auth\_token, timestamp expire\_dt);

integer LockRow(text a\_table\_name, text a\_row\_key, text an\_auth\_token, timestamp expire\_dt);

integer LockRow(text a\_table\_name, text a\_row\_key, text an\_auth\_token);

**描述**

对于一个表中具体的行加锁或者说加认证。参数expire默认的过期时间是当前时间的一个小时之后过期。

如果成功返回1，否则返回0（表示已经加了锁）

可用版本: 1.1.3

**样例**

SELECT LockRow(’public’, ’towns’, ’2’, ’joey’);

LockRow

-------

1

--Joey has already locked the record and Priscilla is out of luck

SELECT LockRow(’public’, ’towns’, ’2’, ’priscilla’);

LockRow

-------

0

**请参考**

UnlockRows

**8.13.6**

**UnlockRows**

UnlockRows — 根据认证token，移除所有记录的锁，返回被释放的记录数

**用法**

integer UnlockRows(text auth\_token);

PostGIS 2.2.0dev 开发手册

358 / 686

**描述**

根据认证token，移除所有记录的锁，返回被释放的记录数

可用版本: 1.1.3

**样例**

SELECT LockRow(’towns’, ’353’, ’priscilla’);

SELECT LockRow(’towns’, ’2’, ’priscilla’);

SELECT UnLockRows(’priscilla’);

UnLockRows

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

2

**请参考**

LockRow

**8.14**

**8.14.1**

**其他五花八门的函数**

**ST\_Accum**

ST\_Accum — 聚合函数，返回输入几何对象形成的数组

**用法**

geometry[] ST\_Accum(geometry set geomﬁeld);

**描述**

聚合函数，返回输入几何对象形成的数组

版本提升: 2.0.0 版本支持Polyhedral Surface类型、Triangle类型、TIN类型.

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

该函数支持CircularString和Curve几何类型对象

该函数支持 Polyhedral Surface类型几何对象.

该函数支持 Triangles 和 Triangulated Irregular Network Surfaces (TIN)类型.

**样例**

SELECT (ST\_Accum(the\_geom)) As all\_em, ST\_AsText((ST\_Accum(the\_geom))[1]) As grabone,

(ST\_Accum(the\_geom))[2:4] as grab\_rest

FROM (SELECT ST\_MakePoint(a\*CAST(random()\*10 As integer), a\*CAST(random()\*10 As

integer), a\*CAST(random()\*10 As integer)) As the\_geom

FROM generate\_series(1,4) a) As foo;

←



PostGIS 2.2.0dev 开发手册

359 / 686

all\_em|grabone

| grab\_rest

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

{0101000080000000000000144000000000000024400000000000001040:

0101000080000000000

00018400000000000002C400000000000003040:

0101000080000000000000354000000000000038400000000000001840:

010100008000000000000040400000000000003C400000000000003040} |

POINT(5 10) | {010100008000000000000018400000000000002C400000000000003040:

0101000080000000000000354000000000000038400000000000001840:

010100008000000000000040400000000000003C400000000000003040}

(1 row)

**请参考**

ST\_Collect

**8.14.2**

**Box2D**

Box2D — 返回一个几何对象的最大扩展BOX2D对象

**用法**

box2d Box2D(geometry geomA);

**描述**

返回一个几何对象的最大扩展BOX2D对象

版本提升: 2.0.0 版本支持Polyhedral Surface类型、Triangle类型、TIN类型.

该函数支持CircularString和Curve几何类型对象

该函数支持 Polyhedral Surface类型几何对象.

该函数支持 Triangles 和 Triangulated Irregular Network Surfaces (TIN)类型.

**样例**

SELECT Box2D(ST\_GeomFromText(’LINESTRING(1 2, 3 4, 5 6)’));

box2d

---------

BOX(1 2,5 6)

SELECT Box2D(ST\_GeomFromText(’CIRCULARSTRING(220268 150415,220227 150505,220227 150406)’) ←

);

box2d

--------

BOX(220186.984375 150406,220288.25 150506.140625)



PostGIS 2.2.0dev 开发手册

360 / 686

**请参考**

Box3D, ST\_GeomFromText

**8.14.3**

**Box3D**

Box3D —返回一个几何对象的最大扩展BOX3D对象

**用法**

box3d Box3D(geometry geomA);

**描述**

返回一个几何对象的最大扩展BOX3D对象

版本提升: 2.0.0 版本支持Polyhedral Surface类型、Triangle类型、TIN类型.

该函数支持CircularString和Curve几何类型对象

该函数支持 Polyhedral Surface类型几何对象.

该函数支持 Triangles 和 Triangulated Irregular Network Surfaces (TIN)类型.

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

**样例**

SELECT Box3D(ST\_GeomFromEWKT(’LINESTRING(1 2 3, 3 4 5, 5 6 5)’));

Box3d

---------

BOX3D(1 2 3,5 6 5)

SELECT Box3D(ST\_GeomFromEWKT(’CIRCULARSTRING(220268 150415 1,220227 150505 1,220227

150406 1)’));

Box3d

--------

BOX3D(220227 150406 1,220268 150415 1)

**请参考**

Box2D, ST\_GeomFromEWKT

←

**8.14.4**

**ST\_EstimatedExtent**

ST\_EstimatedExtent — 返回一个给定的空间类型的估算box2d对象。该估算是基于对几何类型列的统计。如果没有指定schema，使用当前current\_schema()对应的值

**用法**

box2d ST\_EstimatedExtent(text schema\_name, text table\_name, text geocolumn\_name);

box2d ST\_EstimatedExtent(text table\_name, text geocolumn\_name);



PostGIS 2.2.0dev 开发手册

361 / 686

**描述**

返回一个给定的空间类型的估算box2d对象。该估算是基于对几何类型列的统计。如果没有指定schema，使用当前current\_schema()对应的值。

对于 PostgreSQL版本>=8.0.0，该函数统计是基于VACUUM ANALYZE，统计的长度大概是实际的95%

**注意**

如果没有统计（空表或者没有调用ANALYZE进程），该函数会返回NULL，在1.5.4版本之前，这种情况会抛出一个异常

对于PostgreSQL<8.0.0 统计是基于update\_geometry\_stats() 的结果，返回的结果是精确的。

可用版本: 1.0.0

版本变更: 2.1.0 2.0.x版本之前，该函数被称作ST\_Estimated\_Extent.

该函数支持CircularString和Curve几何类型对象

**样例**

SELECT ST\_EstimatedExtent(’ny’, ’edges’, ’the\_geom’);

--result--

BOX(-8877653 4912316,-8010225.5 5589284)

SELECT ST\_EstimatedExtent(’feature\_poly’, ’the\_geom’);

--result--

BOX(-124.659652709961 24.6830825805664,-67.7798080444336 49.0012092590332)

**请参考**

ST\_Extent

**8.14.5**

**ST\_Expand**

ST\_Expand — 根据输入几何对象的bounding box返回一个向各个方向扩展的bounding box。输出对象的坐标精度使用双精度

**用法**

geometry ST\_Expand(geometry g1, ﬂoat units\_to\_expand);

box2d ST\_Expand(box2d g1, ﬂoat units\_to\_expand);

box3d ST\_Expand(box3d g1, ﬂoat units\_to\_expand);

**描述**

根据输入几何对象的bounding box返回一个向各个方向扩展的bounding box，输出对象的坐标精度使用双精精度。对于距离查询或者bounding box 查询很有用的，因为它可以添加一个索引来对进行查询过滤。

该函数有3种形式。输入对象是geometry类型的，会返回一个表示bounding box的POLYGON类型对象，是最常用的一种形式



PostGIS 2.2.0dev 开发手册

362 / 686

函数ST\_Expand 在概念上和函数ST\_Buffer 很类似，但区别是对几何对象进行各个方向扩展时候，函数ST\_Expand 是扩展bounding box所对应的x，y，z坐标一个指定的单位

单位按照SRID所对应空间参考系所规定的单位

**注意**

在1.3之前，函数ST\_Expand通常用于在距离查询中，让索引可用。例如如下查询the\_geom && ST\_Expand(’POINT(10 20)’, 10) AND ST\_Distance(the\_geom, ’POINT(10 20)’) < 10 Post 1.2，该查询用ST\_DWithin 构建的查询更简单

**注意**

可用版本: 1.5.0，输出使用双精度，而不是float4（单精度）坐标

版本提升: 2.0.0 版本支持Polyhedral Surface类型、Triangle类型、TIN类型.

该函数支持 Polyhedral Surface类型几何对象.

该函数支持 Triangles 和 Triangulated Irregular Network Surfaces (TIN)类型.

**样例**

**注意**

下面的样例是使用US National Atlas Equal Area (SRID=2163) 坐标系

--10 meter expanded box around bbox of a linestring

SELECT CAST(ST\_Expand(ST\_GeomFromText(’LINESTRING(2312980 110676,2312923 110701,2312892

←

110714)’, 2163),10) As box2d);

st\_expand

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

BOX(2312882 110666,2312990 110724)

--10 meter expanded 3d box of a 3d box

SELECT ST\_Expand(CAST(’BOX3D(778783 2951741 1,794875 2970042.61545891 10)’ As box3d),10)

st\_expand

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

BOX3D(778773 2951731 -9,794885 2970052.61545891 20)

--10 meter geometry astext rep of a expand box around a point geometry

SELECT ST\_AsEWKT(ST\_Expand(ST\_GeomFromEWKT(’SRID=2163;POINT(2312980 110676)’),10));

st\_asewkt

----------------------------------------------------------------------------------------------- -- ←

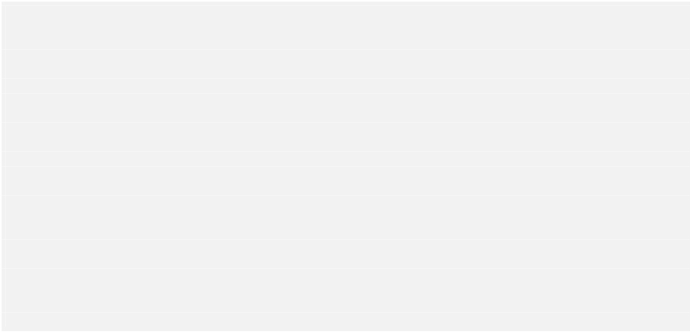
SRID=2163;POLYGON((2312970 110666,2312970 110686,2312990 110686,2312990 110666,2312970

110666))

**请参考**

ST\_AsEWKT, ST\_Buffer, ST\_DWithin, ST\_GeomFromEWKT, ST\_GeomFromText, ST\_SRID

←



PostGIS 2.2.0dev 开发手册

363 / 686

**8.14.6**

**ST\_Extent**

ST\_Extent — 一个返回包含多个geometry对象的bounding box的聚集函数

**用法**

box2d ST\_Extent(geometry set geomﬁeld);

**描述**

一个返回包含多个geometry对象的bounding box的聚集函数。该函数是聚合函数意味着，对于多行的数据，该函数的作用方式会像SQL里面的聚合函数SUM() and AVG() 一样。由于该函数返回bounding box，空间单位以SRID对应的参考系所规定的单位

ST\_Extent is 函数对应Oracle Oracle Spatial/Locator 中的函数SDO\_AGGR\_MBR

**注意**

因为函数ST\_3Dextent返回一个bounding box，但SRID值会被丢掉，使用函数ST\_SetSRID函数重新把SRID值写进该几何对象中去。空间单位以SRID对应的参考系所规定的单位。

**注意**

ST\_Extent 函数返回的bounding box值包含x和y坐标，即便输入对象是(x,y,z)形式的坐标。如果想返回x,y,z形式的3维坐标，使用函数

ST\_3DExtent instead.

**注意**

可用版本: 1.4.0

版本提升: 2.0.0 版本支持Polyhedral Surface类型、Triangle类型、TIN类型.

该函数支持 Polyhedral Surface类型几何对象.

该函数支持 Triangles 和 Triangulated Irregular Network Surfaces (TIN)类型.

**样例**

**注意**

样例 below use Massachusetts State Plane ft (SRID=2249)

SELECT ST\_Extent(the\_geom) as bextent FROM sometable;

st\_bextent

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

BOX(739651.875 2908247.25,794875.8125 2970042.75)



PostGIS 2.2.0dev 开发手册

364 / 686

--Return extent of each category of geometries

SELECT ST\_Extent(the\_geom) as bextent

FROM sometable

GROUP BY category ORDER BY category;

bextent

|

name

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

BOX(778783.5625 2951741.25,794875.8125 2970042.75) | A

BOX(751315.8125 2919164.75,765202.6875 2935417.25) | B

BOX(739651.875 2917394.75,756688.375 2935866) | C

--Force back into a geometry

-- and render the extended text representation of that geometry

SELECT ST\_SetSRID(ST\_Extent(the\_geom),2249) as bextent FROM sometable;

bextent

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

SRID=2249;POLYGON((739651.875 2908247.25,739651.875 2970042.75,794875.8125 2970042.75,

794875.8125 2908247.25,739651.875 2908247.25))

**请参考**

ST\_AsEWKT, ST\_3DExtent, ST\_SetSRID, ST\_SRID

**8.14.7**

**ST\_3DExtent**

ST\_3DExtent —一个返回包含多个geometry对象的3D bounding box的聚集函数

**用法**

box3d ST\_3DExtent(geometry set geomﬁeld);

**描述**

一个返回包含多个geometry对象的3D bounding box（包含Z坐标）的聚集函数。该函数是聚合函数意味着，对于多行的数据，该函数的作用方式会像SQL里面的聚合函数SUM() and AVG() 一样。由于该函数返回bounding box，空间单位以SRID对应的参考系所规定的单位

**注意**

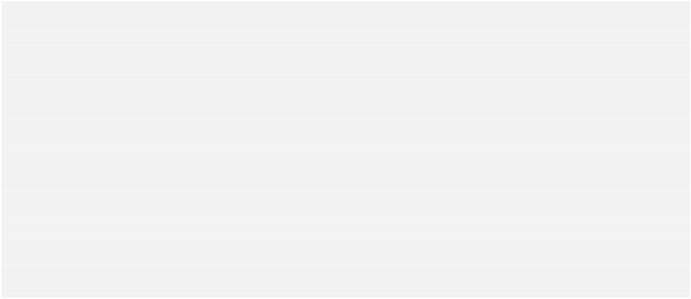
因为函数ST\_3Dextent返回一个bounding box，但SRID值会被丢掉，使用函数ST\_SetSRID函数重新把SRID值写进该几何对象中去。空间单位以SRID对应的参考系所规定的单位

版本提升: 2.0.0 版本支持Polyhedral Surface类型、Triangle类型、TIN类型.

版本变更: 2.0.0 In prior versions this used to be called ST\_Extent3D

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

该函数支持CircularString和Curve几何类型对象



PostGIS 2.2.0dev 开发手册

365 / 686

该函数支持 Polyhedral Surface类型几何对象.

该函数支持 Triangles 和 Triangulated Irregular Network Surfaces (TIN)类型.

**样例**

SELECT ST\_3DExtent(foo.the\_geom) As

FROM (SELECT ST\_MakePoint(x,y,z) As

FROM generate\_series(1,3) As x

CROSS JOIN generate\_series(1,2)

CROSS JOIN generate\_series(0,2)

b3extent

the\_geom

As y

As Z) As foo;

b3extent

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

BOX3D(1 1 0,3 2 2)

--Get the extent of various elevated circular strings

SELECT ST\_3DExtent(foo.the\_geom) As b3extent

FROM (SELECT ST\_Translate(ST\_Force\_3DZ(ST\_LineToCurve(ST\_Buffer(ST\_MakePoint(x,y),1))),0,0, ←

z) As the\_geom

FROM generate\_series(1,3) As x

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

CROSS JOIN generate\_series(0,2) As Z) As foo;

b3extent

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

BOX3D(1 0 0,4 2 2)

**请参考**

ST\_Extent, ST\_Force3DZ

**8.14.8**

**Find\_SRID**

Find\_SRID —该函数的用法是ﬁnd\_srid(a\_db\_schema, a\_table, a\_column)，函数的返回值一个指定列的整型SRID值，该函数实际上是通过查询GEOMETRY\_COLUMNS 表得到的。

**用法**

integer Find\_SRID(varchar a\_schema\_name, varchar a\_table\_name, varchar a\_geomﬁeld\_name);

**描述**

该函数的用法是ﬁnd\_srid(a\_db\_schema, a\_table, a\_column)，函数的返回值一个指定列的整型SRID值，该函数实际上是通过查询GEOMETRY\_COLUMNS 表得到的。如果geometry类型列没有通过函数AddGeometryColumns() 来正确地添加，该函数无法对该列使用

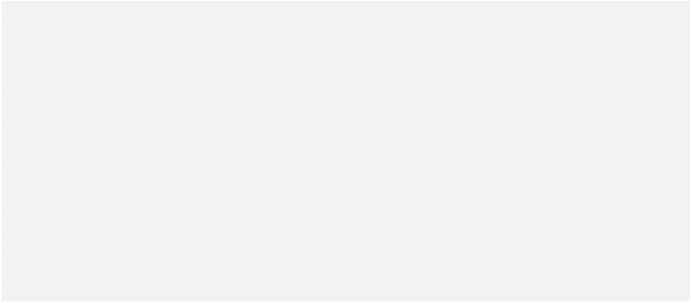
**样例**

SELECT Find\_SRID(’public’, ’tiger\_us\_state\_2007’, ’the\_geom\_4269’);

find\_srid

----------

4269



PostGIS 2.2.0dev 开发手册

366 / 686

**请参考**

ST\_SRID

**8.14.9**

**ST\_Mem\_Size**

ST\_Mem\_Size — 返回一个几何对象占用的空间（以byte字节为单位）

**用法**

integer ST\_Mem\_Size(geometry geomA);

**描述**

返回一个几何对象占用的空间（以byte字节为单位）

该函数和PostgreSQL内置的函数pg\_size\_pretty, pg\_relation\_size, pg\_total\_relation\_size一样非常值得称赞

**注意**

函数pg\_relation\_size给出一个表所占用空间比ST\_Mem\_Size要少，这是因为pg\_relation\_size不会把TOAST类型表中存储大型几何对象

的字段算进去。而函数pg\_total\_relation\_size –会把所有的普通表、toast类型表和索引都计算进去。（译者注：Toast是一个名字缩写，

全写是The OverSized Attribute Storage Technique，即超尺寸字段存储技术，顾名思义，是说超长字段在Postgres的一个存储方式。

Postgres采用的存储默认是每个页面存储固定8Kb大小的数据，并且元组不允许跨页面存储，所以并不能直接存储大字段数据。

Toast就是为此应运而生，它会将大字段值压缩或者分散为多个物理行来存储。对于用户来说完全不用关注这一技术实现，完全是透明的。）

这个函数支持3D对象，并且不会删除z坐标（译者注：3D对象会有3个坐标，x,y,z）.

该函数支持CircularString和Curve几何类型对象

该函数支持 Polyhedral Surface类型几何对象.

该函数支持 Triangles 和 Triangulated Irregular Network Surfaces (TIN)类型.

**样例**

--Return how much byte space Boston takes up

in our Mass data set

SELECT pg\_size\_pretty(SUM(ST\_Mem\_Size(the\_geom))) as totgeomsum,

pg\_size\_pretty(SUM(CASE WHEN town = ’BOSTON’ THEN st\_mem\_size(the\_geom) ELSE 0 END)) As

bossum,

CAST(SUM(CASE WHEN town = ’BOSTON’ THEN st\_mem\_size(the\_geom) ELSE 0 END)\*1.00 /

SUM(st\_mem\_size(the\_geom))\*100 As numeric(10,2)) As perbos

FROM towns;

←

totgeomsum

bossum

perbos

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

1522 kB

30 kB 1.99

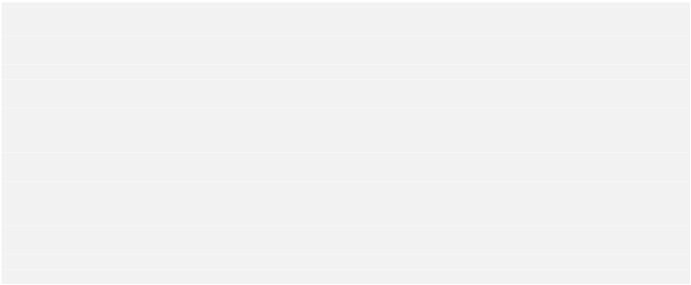
SELECT ST\_Mem\_Size(ST\_GeomFromText(’CIRCULARSTRING(220268 150415,220227 150505,220227

150406)’));

---

73

←



PostGIS 2.2.0dev 开发手册

367 / 686

--What percentage of our table is taken up by just the geometry

SELECT pg\_total\_relation\_size(’public.neighborhoods’) As fulltable\_size, sum(ST\_Mem\_Size( ←

the\_geom)) As geomsize,

sum(ST\_Mem\_Size(the\_geom))\*1.00/pg\_total\_relation\_size(’public.neighborhoods’)\*100 As

pergeom

FROM neighborhoods;

←

fulltable\_size geomsize

pergeom

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

262144 96238 36.71188354492187500000

**请参考**

**8.14.10**

**ST\_Point\_Inside\_Circle**

ST\_Point\_Inside\_Circle — 返回一个点是否在一个Circle对象内，Circle定义语法是中心点的坐标和半径

**用法**

boolean ST\_Point\_Inside\_Circle(geometry a\_point, ﬂoat center\_x, ﬂoat center\_y, ﬂoat radius);

**描述**

该函数的用法是st\_point\_inside\_circle(<geometry>,<circle\_center\_x>,<circle\_center\_y>,<radius>)。如果geometry对象是一个点，并且在一个一个圆内，返回TRUE，否则返回false

**注意**

该函数就像函数名所体现的那样，只支持Point类型对象

**样例**

SELECT ST\_Point\_Inside\_Circle(ST\_Point(1,2), 0.5, 2, 3);

st\_point\_inside\_circle

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

t

**请参考**

ST\_DWithin

**8.15**

**特殊用途函数**

这些函数很少会被用到，如果你的数据出现损坏情况，那么可能会用到这些函数。这些函数是用来解决一些蛋疼的问题的，正常情况下，不会用到这些函数。

**8.15.1**

**PostGIS\_AddBBox**

PostGIS\_AddBBox — 给一个几何对象添加bounding box



PostGIS 2.2.0dev 开发手册

368 / 686

**用法**

geometry PostGIS\_AddBBox(geometry geomA);

**描述**

给一个几何对象添加bounding box.这会让查询更快，但是会增加几何对象的大小（译者注：bounding box就是一个以给定的几何对象为中心的一个最小边界框，所以看起来将该几何对象扩大了）

**注意**

Bounding boxes are automatically added to geometries so in general this is not needed unless the generated bounding

box somehow becomes corrupted or you have an old install that is lacking bounding boxes. Then you need to drop the

old and readd.

该函数支持CircularString和Curve几何类型对象

**样例**

UPDATE sometable

SET the\_geom =

PostGIS\_AddBBox(the\_geom)

WHERE PostGIS\_HasBBox(the\_geom) = false;

**请参考**

PostGIS\_DropBBox, PostGIS\_HasBBox

**8.15.2**

**PostGIS\_DropBBox**

PostGIS\_DropBBox —删除一个geometry类型对象的bounding box 缓存

**用法**

geometry PostGIS\_DropBBox(geometry geomA);

**描述**

删除一个geometry类型对象的bounding box 缓存。这种操作会减小geometry类型对象的大小，但是会让查询变慢。该函数也用于丢弃一个损坏的bounding box。所谓的损坏的bounding box是当函数ST\_Intersects 以及其他关系查询中应该返回true 却被省略掉的几何对象

**注意**

bounding box通常在查询中会被自动添加以便提高查询速度，因此这个函数通常不会被使用，除非因为未知原因bounding box被损坏了或者你安装了缺少bounding box特性的旧的PostGIS版本。在这种情况下，你需要丢弃旧的bounding box，并重新生成bounding box。这种损坏bounding box的情况在PostgreSQL 8.3到8.3.6版本均有，具体情况是当一个几何对象变更或者升级到没有导出及重新装载到新版本时候，原本正确的bounding box不会被重新计算。因此使用者可以手动使用如下方式重建bounding box或者dump 出数据，再reload

该函数支持CircularString和Curve几何类型对象



PostGIS 2.2.0dev 开发手册

369 / 686

**样例**

--This example drops bounding boxes where the cached box is not correct

--The force to ST\_AsBinary before applying Box2D forces a recalculation of the box,

and Box2D applied to the table geometry always

-- returns the cached bounding box.

UPDATE sometable

←

SET the\_geom =

PostGIS\_DropBBox(the\_geom)

WHERE Not (Box2D(ST\_AsBinary(the\_geom)) = Box2D(the\_geom));

UPDATE sometable

SET the\_geom =

PostGIS\_AddBBox(the\_geom)

WHERE Not PostGIS\_HasBBOX(the\_geom);

**请参考**

PostGIS\_AddBBox, PostGIS\_HasBBox, Box2D

**8.15.3**

**PostGIS\_HasBBox**

PostGIS\_HasBBox —如果该几何对象的bounding box已被缓存，则返回TRUE，否则返回FALSE.

**用法**

boolean PostGIS\_HasBBox(geometry geomA);

**描述**

如果该几何对象的bounding box已被缓存，则返回TRUE，否则返回FALSE，使用函数PostGIS\_AddBBox和 PostGIS\_DropBBox来控制缓存

该函数支持CircularString和Curve几何类型对象

**样例**

SELECT the\_geom

FROM sometable WHERE PostGIS\_HasBBox(the\_geom) = false;

**请参考**

PostGIS\_AddBBox, PostGIS\_DropBBox

