PostGIS 3.4.0beta1 Manual

Contents

Abstract

PostGIS é uma extensão para o sistema de banco de dados objeto-relacional PostgreSQL que permite que objetos SIG (Sistema de Informação Geográfica) sejam armazenados em banco de dados. O PostGIS inclui suporte a índices espaciais baseado em GiST R-Tree, e funções para analise e processamento de objetos SIG.

Este é o manual para a versão 3.4.0beta1

Chapter 1

Introdução

PostGIS is a spatial extension for the PostgreSQL relational database that was created by Refractions Research Inc, as a spatial database technology research project. Refractions is a GIS and database consulting company in Victoria, British Columbia, Canada, specializing in data integration and custom software development.

PostGIS is now a project of the OSGeo Foundation and is developed and funded by many FOSS4G developers and organizations all over the world that gain great benefit from its functionality and versatility.

The PostGIS project development group plans on supporting and enhancing PostGIS to better support a range of important GIS functionality in the areas of OGC and SQL/MM spatial standards, advanced topological constructs (coverages, surfaces, networks), data source for desktop user interface tools for viewing and editing GIS data, and web-based access tools.

1.1 Comitê Diretor do Projeto

O Comitê Diretor do Projeto PostGIS (PSC - Project Steering Comitee, em inglês) é responsável pela direção geral, ciclos de lançamento, documentação e os esforços para o projeto. Além disso, o comitê dá suporte ao usuário comum, aceita e aprova novas melhorias da comunidade e vota em questões diversas envolvendo o PostGIS, como por exemplo, uma permissão de commit direta, novos membros do comitê e mudanças significativas da API (Application Programming Interface).

Raúl Marín Rodríguez MVT support, Bug fixing, Performance and stability improvements, GitHub curation, alignment of PostGIS with PostgreSQL releases

Regina Obe Buildbot Maintenance, Windows production and experimental builds, documentation, alignment of PostGIS with PostgreSQL releases, X3D support, TIGER geocoder support, management functions.

Darafei Praliaskouski Index improvements, bug fixing and geometry/geography function improvements, SFCGAL, raster, GitHub curation, and bot maintenance.

Paul Ramsey (Presidente) Co-founder of PostGIS project. General bug fixing, geography support, geography and geometry index support (2D, 3D, nD index and anything spatial index), underlying geometry internal structures, GEOS functionality integration and alignment with GEOS releases, alignment of PostGIS with PostgreSQL releases, loader/dumper, and Shapefile GUI loader.

Sandro Santilli Bug fixes and maintenance, buildbot maintenance, git mirror management, management functions, integration of new GEOS functionality and alignment with GEOS releases, topology support, and raster framework and low level API functions.

1.2 Contribuidores Núclero Atuais

Nicklas Avén Melhorias em funções de distância (incluindo suporte a distância 3D e funções de relacionamento), Tiny WKB (TWKB) (em desenvolvimento) e suporte ao usuário geral.

Dan Baston Geometry clustering function additions, other geometry algorithm enhancements, GEOS enhancements and general user support

Martin Davis GEOS enhancements and documentation

Björn Harrtell MapBox Vector Tile and GeoBuf functions. Gogs testing and GitLab experimentation.

Aliaksandr Kalenik Geometry Processing, PostgreSQL gist, general bug fixing

1.3 Contribuidores Núclero Passado

Bborie Park Prior PSC Member. Raster development, integration with GDAL, raster loader, user support, general bug fixing, testing on various OS (Slackware, Mac, Windows, and more)

Mark Cave-Ayland Prior PSC Member. Coordinated bug fixing and maintenance effort, spatial index selectivity and binding, loader/dumper, and Shapefile GUI Loader, integration of new and new function enhancements.

Jorge Arévalo Desenvolvimento Raster, suporte do driver GDAL e importador.

Olivier Courtin Funções para entrada e saída de XML (KML, GML)/GeoJSon, suporte a 3D e correção de bugs.

Chris Hodgson Antigo membro do comitê. Desenvolvimento em geral, manutenção do website e buildbot, gerente da incubação na OSGeo.

Mateusz Loskot Suporte CMake para o PostGIS, criou o carregador raster original em Python e funções de baixo nível da API raster

Kevin Neufeld Ex PSC. Documentação e suporte a ferramentas de documentação, suporte e manutenção do builbot, suporte avançado de usuários em listas de discussão e melhorias em funções do PostGIS

Dave Blasby Desenvolvedor original e co-fundador do PostGIS. Dave escreveu os objetos do servidor, chamadas de índices e muitas das funcionalidades analíticas presentes no servidor.

Jeff Lounsbury Desenvolvedor original do importador/exportador de shapefiles. Atual representante do Dono do Projeto.

Mark Leslie Manutenção e desenvolvimento de funções do núcleo. Melhorias para o suporte a curvas e no importador GUI.

Pierre Racine Architect of PostGIS raster implementation. Raster overall architecture, prototyping, programming support

David Zwarg Desenvolvimento raster (funções analíticas de álgebra de mapas)

1.4 Outros Contribuidores

Contribuidores Individuais

Alex Bodnaru
Alex Mayrhofer
Andrea Peri
Andreas Forø Tollefsen
Andreas Neumann
Andrew Gierth
Anne Ghisla
Antoine Bajolet
Arthur Lesuisse
Artur Zakirov
Barbara Phillipot
Ben Jubb
Bernhard Reiter
Björn Esser
Brian Hamlin
Bruce Rindahl
Bruno Wolff III
Bryce L. Nordgren
Carl Anderson
Charlie Savage
Christian Schroeder
Christoph Berg
Christoph Moench-Tegeder
Dane Springmeyer
Daryl Herzmann
Dave Fuhry
David Zwarg
David Zwarg
David Zwarg
Dmitry Vasilyev
Eduin Carrillo
Esteban Zimanyi
Eugene Antimirov
Even Rouault
Florian Weimer
Frank Warmerdam
George Silva
Gerald Fenoy
Gino Lucrezi
Greg Troxel
Guillaume Lelarge
Giuseppe Broccolo
Han Wang
Haribabu Kommi
Havard Tveite
IIDA Tetsushi
Ingvild Nystuen
Jackie Leng
James Marca
Jan Katins
Jason Smith
Jeff Adams
Jim Jones
Joe Conway
Jonne Savolainen
Jose Carlos Martinez Llari
Jörg Habenicht
Julien Rouhaud
Kashif Rasul
Klaus Foerster
Kris Jurka
Laurenz Albe
Lars Roessiger
Leo Hsu
Loïc Bartoletti
Loic Dachary
Luca S. Percich
Lucas C. Villa Real
Maria Arias de Reyna
Marc Ducobu
Mark Sondheim
Markus Schaber
Markus Wanner
Matt Amos
Matt Bretl
Matthias Bay
Maxime Guillaud
Maxime van Noppen
Michael Fuhr
Mike Toews
Nathan Wagner
Nathaniel Clay
Nikita Shulga
Norman Vine
Patricia Tozer
Rafal Magda
Ralph Mason
Rémi Cura
Richard Greenwood
Robert Coup
Roger Crew
Ron Mayer
Sebastiaan Couwenberg
Sergei Shoulbakov
Sergey Fedoseev
Shinichi Sugiyama
Shoaib Burq
Silvio Grosso
Stefan Corneliu Petrea
Steffen Macke
Stepan Kuzmin
Stephen Frost
Steven Ottens
Talha Rizwan
Teramoto Ikuhiro
Tom Glancy
Tom van Tilburg
Victor Collod
Vincent Bre
Vincent Mora
Vincent Picavet
Volf Tomáš

Patrocinadores corporativos Estas são entidades corporativas que contribuiram com horas home, hospedagem ou suporte monetário direto ao projeto PostGIS

- Aiven
- Arrival 3D
- Associazione Italiana per l'Informazione Geografica Libera (GFOSS.it)
- AusVet
- Avencia
- Azavea
- Boundless
- Cadcorp
- Camptocamp
- Carto
- Crunchy Data
- City of Boston (DND)

- City of Helsinki
- Clever Elephant Solutions
- Cooperativa Alveo
- Deimos Space
- Faunalia
- Geographic Data BC
- Hunter Systems Group
- ISciences, LLC
- Kontur
- Lidwala Consulting Engineers
- LISAsoft
- Logical Tracking & Tracing International AG
- Maponics
- Michigan Tech Research Institute
- Natural Resources Canada
- Norwegian Forest and Landscape Institue
- Norwegian Institute of Bioeconomy Research (NIBIO)
- OSGeo
- Oslandia
- Palantir Technologies
- Paragon Corporation
- R3 GIS
- Refractions Research
- Regione Toscana - SITA
- Safe Software
- Sirius Corporation plc
- Stadt Uster
- UC Davis Center for Vectorborne Diseases
- Université Laval
- U.S. Department of State (HIU)
- Zonar Systems

Campanhas de financiamento coletivo Crowd funding campaigns - Campanhas de financiamento de multidões são campanhas que executamos para obter os recursos que queremos para financiar um projeto por um grande número de pessoas. Cada campanha é especificamente focada em um recurso ou conjunto de recursos específicos. Cada patrocinador utiliza uma fração pequena do financiamento necessário e com o número suficiente de pessoas / organizações contribuindo, temos os fundos para pagar o trabalho que vai ajudar muitos. Se você tiver uma idéia de um recurso que você acha que muitos outros estariam dispostos a co-financiar, por favor, postar para o newsgroup PostGIS suas ideias, e juntos podemos fazer isso acontecer.

A versão 2.0.0 foi a primeira em que testamos esta estratégia. Utilizamos o PledgeBank e conseguimos realizar duas campanhas bem sucedidas.

postgistopology - 10 patrocinadores, cada um contribuiu com USD $250,00 para a construção da função toTopoGeometry e melhorias gerais no suporte a topologia da versão 2.0.0. Aconteceu!

postgis64windows - 20 patrocinadores, contribuiram com $100 USD cada, para pagar para a compilação do PostGIS no Windows 64bits. Aconteceu. Agora temos uma versão 64-bits do PostGIS 2.0.1 disponível na PostgreSQL Stack Builder.

Bibliotecas importantes The GEOS geometry operations library

The GDAL Geospatial Data Abstraction Library used to power much of the raster functionality introduced in PostGIS 2. In kind, improvements needed in GDAL to support PostGIS are contributed back to the GDAL project.

The PROJ cartographic projection library

Por último, mas não menos importante, o PostgreSQL DBMS, o gigante sobre qual o PostGIS se apóia. Muito da velocidade e flexibilidade do PostGIS não seria possível sem a extensibilidade, um grande analisador de consultas, índice GIST e uma variedade de funcionalidades SQL dadas pelos PostgreSQl.

Chapter 2

Instalação do PostGIS

Este capítulo detalha os passos necessários para instalar o PostGIS.

2.1 Versão Reduzida

Para compilar, assumindo que você tem todas as dependências em seu caminho de busca (search path):

```
tar -xvfz postgis-3.4.0beta1.tar.gz
cd postgis-3.4.0beta1
./configure
make
make install
```

Assim que o PostGIS esteja instalado, ele precisa ser habilitado em cada banco de dados que você deseje utilizá-lo.

2.2 Compilando e instalando da fonte: detalhado

Note
Muitos sistemas operacionas agora incluem pacotes pré-compilados para PostgreSQL / PostGIS. Em muitos casos, a compilação só é necessário se você quiser as versões ponta ou você é um mantenedor do pacote.
This section includes general compilation instructions, if you are compiling for Windows etc or another OS, you may find additional more detailed help at PostGIS User contributed compile guides and PostGIS Dev Wiki.
Pre-Built Packages for various OS are listed in PostGIS Pre-built Packages
Se você é um usuário windows, você pode obter builds estáveis via Stackbuilder PostGIS Windows download site
Também builds experimentais para windows são builds lançadadas geramente uma ou duas vezes por semana ou sempre que algo emocionante acontece. Você pode usá-los para experimentar os lançamentos em progresso de PostGIS

The PostGIS module is an extension to the PostgreSQL backend server. As such, PostGIS 3.4.0beta1 *requires* full PostgreSQL server headers access in order to compile. It can be built against PostgreSQL versions 12 - 16. Earlier versions of PostgreSQL are *not* supported.

Refer to the PostgreSQL installation guides if you haven't already installed PostgreSQL. https://www.postgresql.org .

Note

Para funcionalidade da GEOS, quando você instalar o PostgreSQL você pode ter que linkar explicitamente o PostgreSQL contra a biblioteca padrão C++:

```
LDFLAGS=-lstdc++ ./configure [YOUR OPTIONS HERE]
```

Isto é uma forma de contornar as exceções falso-positivas da interação do C++ com ferramentas de desenvolvimento mais antigas. Se você experimentar problemas estranho (backend fechando de forma inesperada ou coisas similares), tente este truque. Isto irá requerir que você compile o PostgreSQL do zero, claro.

Os passos a seguir demonstram a configuração e compilação dos fontes do PostGIS. Eles são escritos para usuários de Linux e não funcionarão em Windows ou Mac.

2.2.1 Obtendo o Fonte

Obtenha o fonte do PostGIS através da seção de downloads do website http://download.osgeo.org/postgis/source/postgis-3.4.0beta1.tar.gz

```
wget http://download.osgeo.org/postgis/source/postgis-3.4.0beta1.tar.gz
tar -xvzf postgis-3.4.0beta1.tar.gz
cd postgis-3.4.0beta1
```

Isto irá criar um diretório chamado `postgis-3.4.0beta1` no diretório de trabalho atual.

Outra alternativa ,é o checkout da fonte do svn repository http://svn.osgeo.org/postgis/trunk/ .

```
git clone https://git.osgeo.org/gitea/postgis/postgis.git postgis
cd postgis
sh autogen.sh
```

Mude para o recém criado `postgis-3.4.0beta1` diretório para continuar a instalação.

```
./configure
```

2.2.2 Instalando pacotes requeridos

PostGIS tem os seguintes requisitos para a construção e uso:

Necessário

- PostgreSQL 12 - 16. A complete installation of PostgreSQL (including server headers) is required. PostgreSQL is available from http://www.postgresql.org .

  For a full PostgreSQL / PostGIS support matrix and PostGIS/GEOS support matrix refer to https://trac.osgeo.org/postgis/wiki/UsersWikiPostgreSQLPostGIS

- Compilador GNU C (`gcc`). Alguns outros compiladores ANSI C podem ser utilizados para compilar o PostGIS, mas nós encontramos menos problemas ao compilar com `gcc`.

- GNU Make (`gmake` ou `make`). Para varios sistemas, GNU `make` é a versão padrão do make. Verifique a versão invocando `make -v`. Outras versões do `make` pode não processar o PostGIS `Makefile` corretamente.

- Proj reprojection library. Proj 6.1 or above is required. The Proj library is used to provide coordinate reprojection support within PostGIS. Proj is available for download from https://proj.org/ .

- GEOS geometry library, version 3.6 or greater, but GEOS 3.11+ is required to take full advantage of all the new functions and features. GEOS is available for download from https://libgeos.org .

- LibXML2, version 2.5.x or higher. LibXML2 is currently used in some imports functions (ST_GeomFromGML and ST_GeomFromKM LibXML2 is available for download from https://gitlab.gnome.org/GNOME/libxml2/-/releases.
- JSON-C, versão 0.9 ou maior. JSON-C é atualmente utilizado para importar GeoJSON através da função ST_GeomFromGeoJson. JSON-C está disponível para download em https://github.com/json-c/json-c/releases/.
- GDAL, version 2+ is required 3+ is preferred. This is required for raster support. https://gdal.org/download.html.
- Este parâmetro está atualmente sem funcionalidade, já que o pacote somente irá instalar na localização do PostgreSQL. Visite http://trac.osgeo.org/postgis/ticket/635 para acompanhar este bug.

Opcional

- Certifique-se também de ativar os dispositivos que deseja usar como está descrito em Section 2.1.
- GTK (requer GTK+2.0, 2.8+) para compilar o shp2pgsql-gui para formar o carregador de arquivo. http://www.gtk.org/ .
- SFCGAL, version 1.3.1 (or higher), 1.4.1 or higher is recommended and required to be able to use all functionality. SFCGAL can be used to provide additional 2D and 3D advanced analysis functions to PostGIS cf Section 7.21. And also allow to use SFCGAL rather than GEOS for some 2D functions provided by both backends (like ST_Intersection or ST_Area, for instance). A PostgreSQL configuration variable `postgis.backend` allow end user to control which backend he want to use if SFCGAL is installed (GEOS by default). Nota: SFCGAL 1.2 require at least CGAL 4.3 and Boost 1.54 (cf: https://oslandia.gitlab.io/SFCGAL/dev.html) https://gitlab.com/Oslandia/SFCGAL/.
- In order to build the Section 11.1 you will also need PCRE http://www.pcre.org (which generally is already installed on nix systems). Section 11.1 will automatically be built if it detects a PCRE library, or you pass in a valid `--with-pcre-dir=/path/to/p` during configure.
- To enable ST_AsMVT protobuf-c library 1.1.0 or higher (for usage) and the protoc-c compiler (for building) are required. Also, pkg-config is required to verify the correct minimum version of protobuf-c. See protobuf-c. By default, Postgis will use Wagyu to validate MVT polygons faster which requires a c++11 compiler. It will use CXXFLAGS and the same compiler as the PostgreSQL installation. To disable this and use GEOS instead use the `--without-wagyu` during the configure step.
- CUnit (`CUnit`). Isto é necessário para o teste de regressão. http://cunit.sourceforge.net/
- DocBook (`xsltproc`)é necessário para a construção da documentação. Docbook esta disponível em http://www.docbook.org/- .
- DBLatex (`dblatex`) é necessário para a construção da documentação em formato PDF. DBLatex está disponível em http://dblatex.sou .
- ImageMagick (`convert`) é necessário para gerar as imagens usadas na documentação. ImageMagick está disponível em http://www.imagemagick.org/ .

2.2.3 Configuração

Como a maior parte das instalações Linux, o primeiro passo é gerar o Makefile que será utilizado para construção do código fonte. Isto é feito utilizando o script shell

./configure

Sem parâmetros adicionais, este comando tentará automaticamente localizar os componentes necessários e bibliotecas para construção do fonte do PostGIS em seu sistema. Embora esta é a forma comum de uso do **./configure**, o script aceita diversos parâmetros para aqueles que tem as bibliotecas e programas necessários em localizações do sistema operacional que não são padrão.

A lista a seguir mostra apenas os parâmetros comumente utilizados. Para uma lista completa, utilize os parâmetros **--help** ou **--help=short**.

--with-library-minor-version Starting with PostGIS 3.0, the library files generated by default will no longer have the minor version as part of the file name. This means all PostGIS 3 libs will end in `postgis-3`. This was done to make pg_upgrade easier, with downside that you can only install one version PostGIS 3 series in your server. To get the old behavior of file including the minor version: e.g. `postgis-3.0` add this switch to your configure statement.

--prefix=PREFIX Esta é a localização onde as bibliotecas do PostGIS e scripts SQL serão instalados. Por padrão, esta localização é a mesma detectada pela instalação do PostgreSQL.

Caution
Este parâmetro está atualmente sem funcionalidade, já que o pacote somente irá instalar na localização do PostgreSQL. Visite http://trac.osgeo.org/postgis/ticket/635 para acompanhar este bug.

--with-pgconfig=FILE O PostgreSQL oferece um utilitário chamado **pg_config** para habilitar extensões como o PostGIS a localizar a instalação do PostgreSQL. Use o parâmetro (**--with-pgconfig=/path/to/pg_config** para especificar manualmente uma instalação específica do PostgreSQL que será usada pelo PostGIS.

--with-gdalconfig=FILE GDAL, uma biblioteca requerida, provê funcionalidades necessárias para o suporte a raster. Use o comando **gdal-config** para localizar o diretório de instalação da GDAL. Use este parâmetro (**--with-gdalconfig=/path/to/gdal-config**) para manualmente especificar uma instalação em particular da GDAL que o PostGIS irá utilizar.

--with-geosconfig=FILE GEOS é uma biblioteca requerida, dá um utilitário chamado **geos-config** para localizar o diretório de instalação da GEOS. Use este parâmetro (**--with-geosconfig=/path/to/geos-config**) para especificar manualmente uma instalação da GEOS que o PostGIS irá utilizar.

--with-xml2config=FILE LibXML é a biblioteca exigida para fazer os processos GeomFormKML/GML. É encontrada normalmente se você tem o libxml instalado, mas se não tiver ou quiser uma versão específica usada, você precisará apontar o PosGIS para um confi file `xml2-config` específico para ativar as instalações de software para localizar a lista de instalação do LibXML. Use esse parâmetro (**>--with-xml2config=/path/to/xml2-config**) para especificar manualmente uma instalação do LibXML que o PostGIS irá construir contra.

--with-projdir=DIR A Proj4 é uma bilbioteca pra reprojeção de coordenadas, na qual o PostGIS depende. Use este parâmetro (**--with-projdir=/path/to/projdir** para especificar manualmente uma instalação do Proj4 que o PostGIS irá utilizar para compilação.

--with-libiconv=DIR Diretório onde o iconv esta instalado.

--with-jsondir=DIR JSON-C é uma biblioteca MIT-licensed JSON exigida pelo suporte PostGIS ST_GeomFromJSON . Use esse parâmetro (**--with-jsondir=/path/to/jsondir**) para especificar manualmente uma instalação do JSON-C que o PostGIS irá construir contra.

--with-pcredir=DIR PCRE é uma biblioteca BSD-licensed Perl Compatible Regular Expression requerida pela extensão address_standardizer. Use esse parâmetro (**--with-pcredir=/path/to/pcredir**) para especificar manualmente uma instalação do PCRE que o PostGIS irá construir contra.

--with-gui Compile a interface de usuário para importação de dados (requer GTK+2.0). Isto irá criar a ferramenta de interface gráfica shp2pgsql-gui para o utilitário shp2pgsql.

--without-raster Instalar suporte a raster

--without-topology Disable topology support. There is no corresponding library as all logic needed for topology is in postgis-3.4.0beta1 library.

--with-gettext=no Por padrão o PostGIS vai tentar detectar o suporte gettext e compilar com ele, porém se você tiver problemas incompatíveis que causem dano de carregamento, você pode o desabilitar com esse comando. Referir-se ao ticket http://trac.osgeo.org/postgis/ticket/748 para um exemplo de problema resolvido configurando com este. NOTA: que você não está perdendo muito desligando isso. É usado para ajuda internacional para o carregador GUI que ainda não está documentado e permanece experimental.

--with-sfcgal=PATH Por padrão PostGIS não tentará instalar com suporte sfcgal sem esta mudança. `PATH` é um argumento opcional que permite especificar um PATH alternativo para sfcgal-config.

--without-phony-revision Disable updating postgis_revision.h to match current HEAD of the git repository.

Note
Se conseguiu o PostGIS do SVN depósito , o primeiro passo é fazer funcionar o script
./autogen.sh
Este script gera a **configurar** script que na volta é usada para personalizar a instalação do PostGIS.
Se em vez de conseguir o PostGIS como tarball, rodando **./autogen.sh** não é necessariamente como **configurar** já foi gerado.

2.2.4 Construindo

Uma vez que o Makefile tenha sido gerado, compilar o PostGIS é simples como rodar o comando

make

A última linha da saída deve ser "`PostGIS was built successfully. Ready to install.`."

As of PostGIS v1.4.0, all the functions have comments generated from the documentation. If you wish to install these comments into your spatial databases later, run the command which requires docbook. The postgis_comments.sql and other package comments files raster_comments.sql, topology_comments.sql are also packaged in the tar.gz distribution in the doc folder so no need to make comments if installing from the tar ball. Comments are also included as part of the CREATE EXTENSION install.

fazer comentários

Apresentado ao PostGIS 2.0. Isto gera html cheat sheets adequadas para referências rápidas ou para handouts dos estudantes. Exige xsltproc para construir e vai gerar 4 arquivos no folder do documento `topology_cheatsheet.html`, `tiger_geocoder_cheatsheet.html`, `raster_cheatsheet.html`, `postgis_cheatsheet.html`

Você pode baixar alguns pre-construídos disponíveis em html e pdf de PostGIS / PostgreSQL Study Guides

faça anotações

2.2.5 Contruindo extensões PostGIS e implantado-as

As extensões do PostGIS são contruídas e instaladas automaticamente se você estiver usando PostgreSQL 9.1 ou superior.

Se você está compilando do repositório, você precisa de compilar a função de descrições primeiro. Estas são compiladas se você possui o docbook instalado. Você pode também construir manualmente com o comando:

fazer comentários

Construir a documentação não é necessário se você está construindo de uma versão de lançamento no formato tar ball, já que estas são empacotadas pré-construídas com o tar ball.

Se você está construindo o PostGIS contra o PostgreSQL 9.1, as extensão devem ser automaticamente construídas como parte do processo de make. Você pode, contudo, se necessário, construir das pastas de extensões ou copiar os arquivos se você precisar dos mesmos em um servidor diferente.

```
cd extensions
cd postgis
make clean
make
export PGUSER=postgres #overwrite psql variables
make check #to test before install
make install
# to test extensions
make check RUNTESTFLAGS=--extension
```

Note
`make check` uses psql to run tests and as such can use psql environment variables. Common ones useful to override are `PGUSER`,`PGPORT`, and `PGHOST`. Refer to psql environment variables

Os arquivos de extensões sempre serão os mesmos para a mesma versão do PostGIS, independente do Sistemas Operacional, então é fácil copiar os arquivos de extensão de um sistema operacional para outro, desde que você tenha os binários do PostGIS instalados em seus servidores.

Se você deseja instalar as extensões manualmente em um servidor diferente, do seu servidor de desenvolvimento, você precisará copiar os seguintes arquivos da pasta de extensões para a pasta`PostgreSQL /share/extension` da sua instalação do PostgreSQL, bem como os binários necessários para o PostGIS, se você não os tem ainda no servidor de destino.

- Existe arquivos de controle que denotam informações como a versão da extensão a ser instalada, caso não seja especificada. `postgis.control, postgis_topology.control`.
- Todos os arquivos na pasta /sql de cada extensão. Note que estes precisam ser copiados para a raiz da pasta share/extension do PostgreSQL `extensions/postgis/sql/*.sql`, `extensions/postgis_topology/sql/*.sql`

Quando você finalizar este processo, você deverá ver `postgis`, `postgis_topology` como extensões disponíveis no PgAdmin -> Extensões.

Se você está utilizando psql, pode verificar quais estensões estão instaladas executando essa query:

```
SELECT name, default_version,installed_version
FROM pg_available_extensions WHERE name LIKE 'postgis%' or name LIKE 'address%';

             name              | default_version | installed_version
-------------------------------+-----------------+-------------------
 address_standardizer          | 3.4.0beta1      | 3.4.0beta1
 address_standardizer_data_us | 3.4.0beta1      | 3.4.0beta1
 postgis                       | 3.4.0beta1      | 3.4.0beta1
 postgis_raster                | 3.4.0beta1      | 3.4.0beta1
 postgis_sfcgal                | 3.4.0beta1      |
 postgis_tiger_geocoder        | 3.4.0beta1      | 3.4.0beta1
 postgis_topology              | 3.4.0beta1      |
(6 rows)
```

Se você tem a extensão instalada no banco de dados de seu interesse, você a verá mencionada na coluna `installed_version`. Se você não receber nenhum registro de volta, significa que você não tem extensões do PostGIS instaladas no servidor. PgAdmin III 1.14+ também irá lhe dar esta informação na seção `extensions` do navegador de banco de dados e até permitirá o upgrade ou a desinstação utilizando o clique com o botão direito.

Se você tem extensões disponíveis, pode instalar a extensão postgis no seu database escolhido usando a interface da extensão pgAdmin ou rodando esses comandos sql:

```
CREATE EXTENSION postgis;
CREATE EXTENSION postgis_raster;
CREATE EXTENSION postgis_sfcgal;
CREATE EXTENSION fuzzystrmatch; --needed for postgis_tiger_geocoder
--optional used by postgis_tiger_geocoder, or can be used standalone
CREATE EXTENSION address_standardizer;
CREATE EXTENSION address_standardizer_data_us;
CREATE EXTENSION postgis_tiger_geocoder;
CREATE EXTENSION postgis_topology;
```

No psql você pode ver quais versões foram instaladas e qual esquema eles estão instalando.

```
\connect mygisdb
\x
\dx postgis*
```

```
List of installed extensions
-[ RECORD 1 ]-------------------------------------------------
-
Name        | postgis
Version     | 3.4.0beta1
```

```
Schema      | public
Description | PostGIS geometry, geography, and raster spat..
-[ RECORD 2 ]-----------------------------------------------
-
Name        | postgis_tiger_geocoder
Version     | 3.4.0beta1
Schema      | tiger
Description | PostGIS tiger geocoder and reverse geocoder
-[ RECORD 3 ]-----------------------------------------------
-
Name        | postgis_topology
Version     | 3.4.0beta1
Schema      | topology
Description | PostGIS topology spatial types and functions
```

Warning

Extensões table `spatial_ref_sys`, `layer`, `topology` não podem ter o backup feito explicitamente. Só podem ser feito o backup quando a respectiva `postgis` ou `postgis_topology` extensão estiver com o backup feito, o que só acontece quando você faz backup de todo o database. Assim como PostGIS 2.0.1, somente os registros srid não compactados com o PostGIS tem o backup quando há o backup do database, então não faça mudanças nos srids que nós compactamos e espere que suas mudanças estejam lá. Coloque em um em um bilhete se encontrar algum problema. As estruturas de extensões table nunca têm o backup feito desde que elas são criadas com `CREATE EXTENSION` e supostas a serem as mesmas para uma dada versão de uma extensão. Estes comportamentos são construídos na extensão atual do PostSQL, portanto não há nada que possamos fazer a respeito.

Se você instalou 3.4.0beta1 sem usar nosso sistema de extensão maravilhoso, você pode mudar para uma extensão baseada em primeiro atualizando para a última micro versão rodando as scripts atualizadas: `postgis_upgrade_22_minor.sql`,`raster_upgrade_22_minor.sql`,`topology_upgrade_22_minor.sql`.

```
CREATE EXTENSION postgis FROM unpackaged;
CREATE EXTENSION postgis_raster FROM unpackaged;
CREATE EXTENSION postgis_topology FROM unpackaged;
CREATE EXTENSION postgis_tiger_geocoder FROM unpackaged;
```

2.2.6 Testando

Se desejar testar o PostGIS, rode

make check

O comando acima irá rodar através de várias verificações e testes de regressão usando a biblioteca gerada contra o database do PostreSQL atual.

Note

Se você configurou o PostGIS usando o não padronizado PostgreSQL, GEOS, ou Proj4 localizações, talvez você precise adicionar a biblioteca de localizações deles à LD_LIBRARY_PATH variável de ambiente.

Caution

Atualmente, o **faz verificação** confia nas variáveis de ambiente `PATH` e `PGPORT` quando vai fazer as verificações - ele *não* usa a versão PostgreSQL que talvez tenha sido especificada utilizando o parâmetro de configuração **--with-pgconfig**. Portanto, certifique-se que para modificar seu PATH para ser compatível com a instalação do PostgreSQL detectada durante a configuração ou esteja preparado para iminentes aborrecimentos.

If successful, make check will produce the output of almost 500 tests. The results will look similar to the following (numerous lines omitted below):

```
CUnit - A unit testing framework for C - Version 2.1-3
     http://cunit.sourceforge.net/

        .
        .
        .

Run Summary:    Type  Total    Ran Passed Failed Inactive
              suites     44     44    n/a      0        0
               tests    300    300    300      0        0
             asserts   4215   4215   4215      0      n/a
Elapsed time =    0.229 seconds

        .
        .
        .

Running tests

        .
        .
        .

Run tests: 134
Failed: 0


-- if you build with SFCGAL

        .
        .
        .

Running tests

        .
        .
        .

Run tests: 13
Failed: 0

-- if you built with raster support

        .
        .
        .

Run Summary:    Type  Total    Ran Passed Failed Inactive
              suites     12     12    n/a      0        0
               tests     65     65     65      0        0
             asserts  45896  45896  45896      0      n/a



        .
        .
        .

Running tests
```

```
        .
        .
        .

Run tests: 101
Failed: 0

-- topology regress

.
.
.

Running tests

        .
        .
        .

Run tests: 51
Failed: 0

-- if you built --with-gui, you should see this too

     CUnit - A unit testing framework for C - Version 2.1-2
     http://cunit.sourceforge.net/

        .
        .
        .

Run Summary:    Type  Total    Ran Passed Failed Inactive
              suites      2      2    n/a      0        0
               tests      4      4      4      0        0
             asserts      4      4      4      0      n/a
```

As extensões `postgis_tiger_geocoder` and `address_standardizer` , atualmente só suportam o modelo PostgreSQL installcheck. Para testá-los use abaixo. Nota: fazer a instalação não é necessária se você já instalou na raiz do folder code do PostGIS.

Para address_standardizer:

```
cd extensions/address_standardizer
make install
make installcheck
```

Saída deve parecer com:

```
============== dropping database "contrib_regression" ==============
DROP DATABASE
============== creating database "contrib_regression" ==============
CREATE DATABASE
ALTER DATABASE
============== running regression test queries        ==============
test test-init-extensions     ... ok
test test-parseaddress        ... ok
test test-standardize_address_1 ... ok
test test-standardize_address_2 ... ok

====================
 All 4 tests passed.
```

```
====================
```

Para o geocoder tiger, certifique-se que você tem extensões portgis e fuzzystratch disponíveis no seu PostgreSQL. Os testes address_standardizer também irão desprezar se seus postgis construídos com address_standardizer suportar:

```
cd extensions/postgis_tiger_geocoder
make install
make installcheck
```

saída deve parecer com:

```
============== dropping database "contrib_regression" ==============
DROP DATABASE
============== creating database "contrib_regression" ==============
CREATE DATABASE
ALTER DATABASE
============== installing fuzzystrmatch               ==============
CREATE EXTENSION
============== installing postgis                     ==============
CREATE EXTENSION
============== installing postgis_tiger_geocoder      ==============
CREATE EXTENSION
============== installing address_standardizer        ==============
CREATE EXTENSION
============== running regression test queries        ==============
test test-normalize_address   ... ok
test test-pagc_normalize_address ... ok

====================
All 2 tests passed.
====================
```

2.2.7 Instalação

Para instalar o PostGIS, digite

make install

Isso irá copiar a instalação dos arquivos do PostGIS para suas subdireções específicas pelo **--prefix** parâmetro de configuração. Particularmente:

- Os binários do carregador e do dumper estão instalados no `[prefix]/bin`.
- Os arquivos SQL, como `postgis.sql`, estão instalados em `[prefix]/share/contrib`.
- As bibliotecas do PostGIS estão instaladas em `[prefix]/lib`.

Se anteriormente você rodou o comando **make comments** para gerar o arquivo `postgis_comments.sql`, `raster_comments.sql`, instale o arquivo sql para executar

make comments-install

Note

`postgis_comments.sql`, `raster_comments.sql`, `topology_comments.sql` foi separado da construção típica e instalações alvo desde que como isso, veio a dependência extra do **xsltproc**.

2.3 Instalando e usando o padronizador de endereço

A extensão `address_standardizer` era usada para ser um pacote separado que requeria download separado. Do PostGIS 2.2 em diante é compactado. Para mais informações sobre o address_standardize, o que ele faz e como fazer sua configuração, referir-se Section 11.1.

O padronizador pode ser usado em conjunção com a extensão tiger geocoder PostGIS compactada como uma reposição para a Normalize_Address discutida. Para usar como reposição Section 2.4.2. Você também pode utilizar como um building block para seu próprio geocoder ou como padronizar seu endereço para uma comparação de endereços mais fácil.

O padronizador de endereço confia no PCRE que já está instalado na maioria dos sistemas Nix, mas você pode baixar a última versão em: http://www.pcre.org. Se durante Section 2.2.3, o PCRE é encontrado, então a extensão do padronizador de endereço será automaticamente construída. Se você tem um pcre personalizado que queira usar, passe a configurar `--with-pcredir=/path/to/p` onde `/path/to/pcre` é a pasta root para o seu pcre incluso e lista lib.

Para usuários do Windows, o pacote PostGIS 2.1+ já está compactado com o address_standardizer, então não precisa compilar podendo seguir direto para o passo `CREATE EXTENSION`.

Uma vez que instalou, você pode conectar no seu banco de dados e rodar o SQL:

```
CREATE EXTENSION address_standardizer;
```

O teste seguinte não requere tables rules, gaz ou lex.

```
SELECT num, street, city, state, zip
 FROM parse_address('1 Devonshire Place PH301, Boston, MA 02109');
```

Saída deve ser

```
num |         street         |  city  | state |  zip
-----+------------------------+--------+-------+-------
 1   | Devonshire Place PH301 | Boston | MA    | 02109
```

2.4 Installing, Upgrading Tiger Geocoder, and loading data

Extras like Tiger geocoder may not be packaged in your PostGIS distribution. If you are missing the tiger geocoder extension or want a newer version than what your install comes with, then use the `share/extension/postgis_tiger_geocoder.*` files from the packages in Windows Unreleased Versions section for your version of PostgreSQL. Although these packages are for windows, the postgis_tiger_geocoder extension files will work on any OS since the extension is an SQL/plpgsql only extension.

2.4.1 Tiger Geocoder Enabling your PostGIS database

1. These directions assume your PostgreSQL installation already has the postgis_tiger_geocoder extension installed.
2. Conecte ao seu banco de dados vis psql ou pgAdmin ou qualquer outra ferramenta e execute os comandos SQL seguintes. Note que se você está instalando em um banco de dados que já possui o postgis, você não precisa fazer o primeiro passo. Se você já tem a extensão `fuzzystrmatch` instalada, não é preciso fazer o segundo passo também.

   ```
   CREATE EXTENSION postgis;
   CREATE EXTENSION fuzzystrmatch;
   CREATE EXTENSION postgis_tiger_geocoder;
   --this one is optional if you want to use the rules based standardizer ( ↩
       pagc_normalize_address)
   CREATE EXTENSION address_standardizer;
   ```

   Se você já tem a extensão postgis_tiger_geocoder instalada e só quer atualizar para a última versão, execute:

   ```
   ALTER EXTENSION postgis UPDATE;
   ALTER EXTENSION postgis_tiger_geocoder UPDATE;
   ```

Se você fez entradas personalizadas ou alterações nos `tiger.loader_platform` e `tiger.loader_variables`, talvez você precisará atualizar estes.

3. Para confirmar que sua instalação está funcionando corretamente, execute esse sql no seu banco de dados:

```
SELECT na.address, na.streetname,na.streettypeabbrev, na.zip
        FROM normalize_address('1 Devonshire Place, Boston, MA 02109') AS na;
```

Qual deve sair

```
address | streetname | streettypeabbrev |  zip
---------+------------+------------------+-------
           1 | Devonshire | Pl               | 02109
```

4. Criar um novo registro na table `tiger.loader_platform` com os paths dos seus executáveis e servidor.

   Então, por exemplo, para criar um perfil chamado debbie que segue a convenção `sh`, você deveria fazer:

```
INSERT INTO tiger.loader_platform(os, declare_sect, pgbin, wget, unzip_command, psql,  ↩
    path_sep,
                    loader, environ_set_command, county_process_command)
SELECT 'debbie', declare_sect, pgbin, wget, unzip_command, psql, path_sep,
          loader, environ_set_command, county_process_command
  FROM tiger.loader_platform
  WHERE os = 'sh';
```

   E então, edite os paths na coluna *declare_sect* para aqueles que servem ao pg, unzip, shp2pgsql, psql, etc da Debbie.

   Se você não editou essa table `loader_platform`, ela só irá conter casos comuns de localizações de itens e você terá que editar a script gerada depois que ela for gerada.

5. As of PostGIS 2.4.1 the Zip code-5 digit tabulation area `zcta5` load step was revised to load current zcta5 data and is part of the Loader_Generate_Nation_Script when enabled. It is turned off by default because it takes quite a bit of time to load (20 to 60 minutes), takes up quite a bit of disk space, and is not used that often.

   To enable it, do the following:

```
UPDATE tiger.loader_lookuptables SET load = true WHERE table_name = 'zcta520';
```

   If present the Geocode function can use it if a boundary filter is added to limit to just zips in that boundary. The Reverse_Geocode function uses it if the returned address is missing a zip, which often happens with highway reverse geocoding.

6. Criar uma pasta chamada `gisdata` na raiz do servidor ou do seu computador local, se você tem uma rede de conexão rápida com o servidor. Essa pasta está onde os arquivos tiger serão baixados e processados. Se não estiver satisfeito em ter a pasta na raiz do servidor ou, simplesmente, quiser alterar para uma outra pasta para representação, edite o campo `staging_fold` na table `tiger.loader_variables`.

7. Criar uma pasta chamada temp na pasta `gisdata` ou onde designar a `staging_fold`. Esta será a pasta onde o carregador extrai os dados tiger baixados.

8. Then run the Loader_Generate_Nation_Script SQL function make sure to use the name of your custom profile and copy the script to a .sh or .bat file. So for example to build the nation load:

```
psql -c "SELECT Loader_Generate_Nation_Script('debbie')" -d geocoder -tA
> /gisdata/nation_script_load.sh
```

9. Run the generated nation load commandline scripts.

```
cd /gisdata
sh nation_script_load.sh
```

10. After you are done running the nation script, you should have three tables in your `tiger_data` schema and they should be filled with data. Confirm you do by doing the following queries from psql or pgAdmin

```
SELECT count(*) FROM tiger_data.county_all;
```

```
count
-------
  3235
(1 row)
```

```
SELECT count(*) FROM tiger_data.state_all;
```

```
count
-------
    56
(1 row)
```

11. By default the tables corresponding to `bg`, `tract`, `tabblock20` are not loaded. These tables are not used by the geocoder but are used by folks for population statistics. If you wish to load them as part of your state loads, run the following statement to enable them.

```
UPDATE tiger.loader_lookuptables SET load = true WHERE load = false AND lookup_name IN ↩
    ('tract', 'bg', 'tabblock20');
```

    Alternatively you can load just these tables after loading state data using the Loader_Generate_Census_Script

12. For each state you want to load data for, generate a state script Loader_Generate_Script.

Warning
DO NOT Generate the state script until you have already loaded the nation data, because the state script utilizes county list loaded by nation script.

13.
```
psql -c "SELECT Loader_Generate_Script(ARRAY['MA'], 'debbie')" -d geocoder -tA
> /gisdata/ma_load.sh
```

14. Executar as scripts commandlines geradas

```
cd /gisdata
sh ma_load.sh
```

15. Depois que terminar de carregar todos os dados ou estiver parado em um ponto, é bom analisar todas as tiger tables para atualizar as estatísticas (incluindo as estatísticas herdadas)

```
SELECT install_missing_indexes();
vacuum (analyze, verbose) tiger.addr;
vacuum (analyze, verbose) tiger.edges;
vacuum (analyze, verbose) tiger.faces;
vacuum (analyze, verbose) tiger.featnames;
vacuum (analyze, verbose) tiger.place;
vacuum (analyze, verbose) tiger.cousub;
vacuum (analyze, verbose) tiger.county;
vacuum (analyze, verbose) tiger.state;
vacuum (analyze, verbose) tiger.zip_lookup_base;
vacuum (analyze, verbose) tiger.zip_state;
vacuum (analyze, verbose) tiger.zip_state_loc;
```

2.4.2 Usando Padronizador de Endereço com Tiger Geocoder

Uma das maiores queixas das pessoas é a função normalizador de endereços Normalize_Address que normaliza um endereço para preparação antes da geocoding. O normalizador está longe da perfeição e tentar corrigir suas imperfeições demanda um grande número de recursos. Como tal nós integramos com outro projeto que tem um mecanismo de padronizador de endereços muito melhor. Para usar esse novo address_standardizer, você compila a extensão como está descrito em Section 2.3 e instala como uma extensão no seu banco de dados.

Uma vez que você instala essa extensão no mesmo banco de dados que instalou `postgis_tiger_geocoder`, então o Pagc_Normalize_Address pode ser usado ao invés do Normalize_Address. Essa extensão é avessa ao tiger, logo pode ser usada com outras fontes de dados como: endereços internacionais. A extensão tiger geocoder vem compactada com suas próprias versões personalizadas de mesa de regras (`tiger.pagc_rules`) , gaz table (`tiger.pagc_gaz`), e lex table (`tiger.pagc_lex`). Essas você pode adicionar e atualizar para melhorar sua experiência com o padronizador de acordo com suas necessidades.

2.4.3 Required tools for tiger data loading

O carregador processa dados de downloads do site de censo para os respectivos arquivos de nação, solicitações de estados, extrai os arquivos e carrega cada estado para seu grupo separado de state tables. Cada state table herda das tables definidas no esquema `tiger` sendo suficiente apenas para pesquisar aquelas tables para acessar todos os dados e derrubar um conjunto de state tables a qualquer momento usando o Drop_State_Tables_Generate_Script se quiser recarregar um estado ou não precisa de um estado mais.

Para ser capaz de carregar dados, você vai precisar das seguintes ferramentas:

- Uma ferramenta para descompactar os arquivos compactados do site de censo.

  Para Unix como sistemas: executável `unzip` que é instalado, normalmente, na maioria dos Unix como plataformas.

  Para Windows, 7-zip é uma ferramenta comprimir/descomprimir grátis que você pode baixar no http://www.7-zip.org/

- `shp2pgsql` commandline que é instalada por padrão quando você instala o PostGIS.

- `wget` que é uma ferramente grabber da internet, instalada na maioria dos sistemas Unix/Linux.

  Se você está no Windows, você pode obter binários pre compilados do http://gnuwin32.sourceforge.net/packages/wget.htm

If you are upgrading from tiger_2010, you'll need to first generate and run Drop_Nation_Tables_Generate_Script. Before you load any state data, you need to load the nation wide data which you do with Loader_Generate_Nation_Script. Which will generate a loader script for you. Loader_Generate_Nation_Script is a one-time step that should be done for upgrading (from a prior year tiger census data) and for new installs.

Para carregar dados do estado referir-se a Loader_Generate_Script para gerar uma script de dados de carregamento para sua plataforma para os estados que deseja. Note que você pode instalar estes gradativamente. Você não precisa carregar todos os estados de uma só vez. Pode carregá-los à medida que for precisando deles.

Depois que os estados desejados forem carregados, certifique-se de executar o:

```
SELECT install_missing_indexes();
```

como está descrito em Install_Missing_Indexes.

Para testar que está tudo funcionando normalmente, tente executar um geocode em um endereço no seu estado, usando Geocode

2.4.4 Upgrading your Tiger Geocoder Install and Data

First upgrade your postgis_tiger_geocoder extension as follows:

```
ALTER EXTENSION postgis_tiger_geocoder UPDATE;
```

Em seguida, derrube todas as nation tables e carregue as novas. Gere uma drop script com essa declaração SQL, como está detalhado em Drop_Nation_Tables_Generate_Script

```
SELECT drop_nation_tables_generate_script();
```

Execute as declarações geradas drop SQL.

Gere uma script que carrega uma nação com SELECIONAR como está detalhado em Loader_Generate_Nation_Script

Para windows

```
SELECT loader_generate_nation_script('windows');
```

Para unix/linux

```
SELECT loader_generate_nation_script('sh');
```

Refer to Section 2.4.1 for instructions on how to run the generate script. This only needs to be done once.

Note
You can have a mix of different year state tables and can upgrade each state separately. Before you upgrade a state you first need to drop the prior year state tables for that state using Drop_State_Tables_Generate_Script.

2.5 Problemas comuns durante a instalação

Existem várias coisas para averiguar quando a instalação ou atualização não saem como o esperado.

1. Certifique-se que instalou o PostgreSQL 12 ou mais novo e que você está compilando contra a mesma versão da fonte PostgreSQL assim como a versão do PostgreSQL que está sendo executada. Confusões podem acontecer quando sua distribuição (Linux) já instalou o PostgreSQL, ou você instalou o PostgreSQL antes e se esqueceu disso. PostGIS só irá funcionar com o PostgreSQL 12 ou mais novo, e mensagens estranhas e inesperadas de erro aparecerão se você usar uma versão mais antiga. Para verificar a versão PostgreSQL que está sendo executada, conecte ao banco de dados usando psql e faça essa consulta:

   ```
   SELECT version();
   ```

   Se você está usando uma distribuição baseada em RPM, você pode confirmar a existência de pacotes pre instalados utilizando o comando **rpm** como segue: **rpm -qa | grep postgresql**

2. Se sua atualização falhar, certifique-se que você está restaurando em um banco de dados que já possui o PostGIS instalado.

   ```
   SELECT postgis_full_version();
   ```

Também certifique que a configuração detectou a localização e versão corretas do PostgreSQL, da biblioteca do Proj4 e da biblioteca do GEOS.

1. A saída da configuração foi usada para gerar o arquivo `postgis_config.h`. Verifique que as variáveis `POSTGIS_PGSQL_VE` `POSTGIS_PROJ_VERSION` e `POSTGIS_GEOS_VERSION` foram configuradas corretamente.

Chapter 3

PostGIS Administration

3.1 Performance Tuning

Tuning for PostGIS performance is much like tuning for any PostgreSQL workload. The only additional consideration is that geometries and rasters are usually large, so memory-related optimizations generally have more of an impact on PostGIS than other types of PostgreSQL queries.

For general details about optimizing PostgreSQL, refer to Tuning your PostgreSQL Server.

For PostgreSQL 9.4+ configuration can be set at the server level without touching `postgresql.conf` or `postgresql.auto.conf` by using the `ALTER SYSTEM` command.

```
ALTER SYSTEM SET work_mem = '256MB';
-- this forces non-startup configs to take effect for new connections
SELECT pg_reload_conf();
-- show current setting value
-- use SHOW ALL to see all settings
SHOW work_mem;
```

In addition to the Postgres settings, PostGIS has some custom settings which are listed in Section 7.24.

3.1.1 Startup

These settings are configured in `postgresql.conf`:

constraint_exclusion

- Default: partition
- This is generally used for table partitioning. The default for this is set to "partition" which is ideal for PostgreSQL 8.4 and above since it will force the planner to only analyze tables for constraint consideration if they are in an inherited hierarchy and not pay the planner penalty otherwise.

shared_buffers

- Default: ~128MB in PostgreSQL 9.6
- Set to about 25% to 40% of available RAM. On windows you may not be able to set as high.

max_worker_processes This setting is only available for PostgreSQL 9.4+. For PostgreSQL 9.6+ this setting has additional importance in that it controls the max number of processes you can have for parallel queries.

- Default: 8
- Sets the maximum number of background processes that the system can support. This parameter can only be set at server start.

3.1.2 Runtime

work_mem - sets the size of memory used for sort operations and complex queries

- Default: 1-4MB
- Adjust up for large dbs, complex queries, lots of RAM
- Adjust down for many concurrent users or low RAM.
- If you have lots of RAM and few developers:

```
SET work_mem TO '256MB';
```

maintenance_work_mem - the memory size used for VACUUM, CREATE INDEX, etc.

- Default: 16-64MB
- Generally too low - ties up I/O, locks objects while swapping memory
- Recommend 32MB to 1GB on production servers w/lots of RAM, but depends on the # of concurrent users. If you have lots of RAM and few developers:

```
SET maintenance_work_mem TO '1GB';
```

max_parallel_workers_per_gather

This setting is only available for PostgreSQL 9.6+ and will only affect PostGIS 2.3+, since only PostGIS 2.3+ supports parallel queries. If set to higher than 0, then some queries such as those involving relation functions like `ST_Intersects` can use multiple processes and can run more than twice as fast when doing so. If you have a lot of processors to spare, you should change the value of this to as many processors as you have. Also make sure to bump up `max_worker_processes` to at least as high as this number.

- Default: 0
- Sets the maximum number of workers that can be started by a single `Gather` node. Parallel workers are taken from the pool of processes established by `max_worker_processes`. Note that the requested number of workers may not actually be available at run time. If this occurs, the plan will run with fewer workers than expected, which may be inefficient. Setting this value to 0, which is the default, disables parallel query execution.

3.2 Configuring raster support

If you enabled raster support you may want to read below how to properly configure it.

As of PostGIS 2.1.3, out-of-db rasters and all raster drivers are disabled by default. In order to re-enable these, you need to set the following environment variables `POSTGIS_GDAL_ENABLED_DRIVERS` and `POSTGIS_ENABLE_OUTDB_RASTERS` in the server environment. For PostGIS 2.2, you can use the more cross-platform approach of setting the corresponding Section 7.24.

If you want to enable offline raster:

```
POSTGIS_ENABLE_OUTDB_RASTERS=1
```

Any other setting or no setting at all will disable out of db rasters.

In order to enable all GDAL drivers available in your GDAL install, set this environment variable as follows

```
POSTGIS_GDAL_ENABLED_DRIVERS=ENABLE_ALL
```

If you want to only enable specific drivers, set your environment variable as follows:

```
POSTGIS_GDAL_ENABLED_DRIVERS="GTiff PNG JPEG GIF XYZ"
```

Note
If you are on windows, do not quote the driver list

Setting environment variables varies depending on OS. For PostgreSQL installed on Ubuntu or Debian via apt-postgresql, the preferred way is to edit `/etc/postgresql/10/main/environment` where 10 refers to version of PostgreSQL and main refers to the cluster.

On windows, if you are running as a service, you can set via System variables which for Windows 7 you can get to by right-clicking on Computer->Properties Advanced System Settings or in explorer navigating to `Control Panel\All Control Panel Items\System`. Then clicking *Advanced System Settings ->Advanced->Environment Variables* and adding new system variables.

After you set the environment variables, you'll need to restart your PostgreSQL service for the changes to take effect.

3.3 Creating spatial databases

3.3.1 Spatially enable database using EXTENSION

If you are using PostgreSQL 9.1+ and have compiled and installed the extensions/postgis modules, you can turn a database into a spatial one using the EXTENSION mechanism.

Core postgis extension includes geometry, geography, spatial_ref_sys and all the functions and comments. Raster and topology are packaged as a separate extension.

Run the following SQL snippet in the database you want to enable spatially:

```
CREATE EXTENSION IF NOT EXISTS plpgsql;
      CREATE EXTENSION postgis;
      CREATE EXTENSION postgis_raster; -- OPTIONAL
      CREATE EXTENSION postgis_topology; -- OPTIONAL
```

3.3.2 Spatially enable database without using EXTENSION (discouraged)

Note
This is generally only needed if you cannot or don't want to get PostGIS installed in the PostgreSQL extension directory (for example during testing, development or in a restricted environment).

Adding PostGIS objects and function definitions into your database is done by loading the various sql files located in `[prefix]/share/contrib` as specified during the build phase.

The core PostGIS objects (geometry and geography types, and their support functions) are in the `postgis.sql` script. Raster objects are in the `rtpostgis.sql` script. Topology objects are in the `topology.sql` script.

For a complete set of EPSG coordinate system definition identifiers, you can also load the `spatial_ref_sys.sql` definitions file and populate the `spatial_ref_sys` table. This will permit you to perform ST_Transform() operations on geometries.

If you wish to add comments to the PostGIS functions, you can find them in the `postgis_comments.sql` script. Comments can be viewed by simply typing **\dd [function_name]** from a **psql** terminal window.

Run the following Shell commands in your terminal:

```
DB=[yourdatabase]
    SCRIPTSDIR=`pg_config --sharedir`/contrib/postgis-3.3/

    # Core objects
    psql -d ${DB} -f ${SCRIPTSDIR}/postgis.sql
    psql -d ${DB} -f ${SCRIPTSDIR}/spatial_ref_sys.sql
    psql -d ${DB} -f ${SCRIPTSDIR}/postgis_comments.sql # OPTIONAL

    # Raster support (OPTIONAL)
    psql -d ${DB} -f ${SCRIPTSDIR}/rtpostgis.sql
    psql -d ${DB} -f ${SCRIPTSDIR}/raster_comments.sql # OPTIONAL

    # Topology support (OPTIONAL)
    psql -d ${DB} -f ${SCRIPTSDIR}/topology.sql
    psql -d ${DB} -f ${SCRIPTSDIR}/topology_comments.sql # OPTIONAL
```

3.4 Upgrading spatial databases

Upgrading existing spatial databases can be tricky as it requires replacement or introduction of new PostGIS object definitions.

Unfortunately not all definitions can be easily replaced in a live database, so sometimes your best bet is a dump/reload process.

PostGIS provides a SOFT UPGRADE procedure for minor or bugfix releases, and a HARD UPGRADE procedure for major releases.

Before attempting to upgrade PostGIS, it is always worth to backup your data. If you use the -Fc flag to pg_dump you will always be able to restore the dump with a HARD UPGRADE.

3.4.1 Soft upgrade

If you installed your database using extensions, you'll need to upgrade using the extension model as well. If you installed using the old sql script way, you are advised to switch your install to extensions because the script way is no longer supported.

3.4.1.1 Soft Upgrade 9.1+ using extensions

If you originally installed PostGIS with extensions, then you need to upgrade using extensions as well. Doing a minor upgrade with extensions, is fairly painless.

If you are running PostGIS 3 or above, then you should use the PostGIS_Extensions_Upgrade function to upgrade to the latest version you have installed.

```
SELECT postgis_extensions_upgrade();
```

If you are running PostGIS 2.5 or lower, then do the following:

```
ALTER EXTENSION postgis UPDATE;
SELECT postgis_extensions_upgrade();
-- This second call is needed to rebundle postgis_raster extension
SELECT postgis_extensions_upgrade();
```

If you have multiple versions of PostGIS installed, and you don't want to upgrade to the latest, you can explicitly specify the version as follows:

```
ALTER EXTENSION postgis UPDATE TO "3.4.0beta1";
ALTER EXTENSION postgis_topology UPDATE TO "3.4.0beta1";
```

If you get an error notice something like:

```
No migration path defined for ... to 3.4.0beta1
```

Then you'll need to backup your database, create a fresh one as described in Section 3.3.1 and then restore your backup on top of this new database.

If you get a notice message like:

```
Version "3.4.0beta1" of extension "postgis" is already installed
```

Then everything is already up to date and you can safely ignore it. **UNLESS** you're attempting to upgrade from an development version to the next (which doesn't get a new version number); in that case you can append "next" to the version string, and next time you'll need to drop the "next" suffix again:

```
ALTER EXTENSION postgis UPDATE TO "3.4.0beta1next";
ALTER EXTENSION postgis_topology UPDATE TO "3.4.0beta1next";
```

Note
If you installed PostGIS originally without a version specified, you can often skip the reinstallation of postgis extension before restoring since the backup just has `CREATE EXTENSION postgis` and thus picks up the newest latest version during restore.

Note
If you are upgrading PostGIS extension from a version prior to 3.0.0, you will have a new extension *postgis_raster* which you can safely drop, if you don't need raster support. You can drop as follows:

```
DROP EXTENSION postgis_raster;
```

3.4.1.2 Soft Upgrade Pre 9.1+ or without extensions

This section applies only to those who installed PostGIS not using extensions. If you have extensions and try to upgrade with this approach you'll get messages like:

```
can't drop ... because postgis extension depends on it
```

NOTE: if you are moving from PostGIS 1.* to PostGIS 2.* or from PostGIS 2.* prior to r7409, you cannot use this procedure but would rather need to do a HARD UPGRADE.

After compiling and installing (make install) you should find a set of `*_upgrade.sql` files in the installation folders. You can list them all with:

```
ls `pg_config --sharedir`/contrib/postgis-3.4.0beta1/*_upgrade.sql
```

Load them all in turn, starting from `postgis_upgrade.sql`.

```
psql -f postgis_upgrade.sql -d your_spatial_database
```

The same procedure applies to raster, topology and sfcgal extensions, with upgrade files named `rtpostgis_upgrade.sql`, `topology_upgrade.sql` and `sfcgal_upgrade.sql` respectively. If you need them:

```
psql -f rtpostgis_upgrade.sql -d your_spatial_database
```

```
psql -f topology_upgrade.sql -d your_spatial_database
```

```
psql -f sfcgal_upgrade.sql -d your_spatial_database
```

You are advised to switch to an extension based install by running

```
psql -c "SELECT postgis_extensions_upgrade();"
```

Note
If you can't find the `postgis_upgrade.sql` specific for upgrading your version you are using a version too early for a soft upgrade and need to do a HARD UPGRADE.

The PostGIS_Full_Version function should inform you about the need to run this kind of upgrade using a "procs need upgrade" message.

3.4.2 Hard upgrade

By HARD UPGRADE we mean full dump/reload of postgis-enabled databases. You need a HARD UPGRADE when PostGIS objects' internal storage changes or when SOFT UPGRADE is not possible. The Release Notes appendix reports for each version whether you need a dump/reload (HARD UPGRADE) to upgrade.

The dump/reload process is assisted by the postgis_restore.pl script which takes care of skipping from the dump all definitions which belong to PostGIS (including old ones), allowing you to restore your schemas and data into a database with PostGIS installed without getting duplicate symbol errors or bringing forward deprecated objects.

Supplementary instructions for windows users are available at Windows Hard upgrade.

The Procedure is as follows:

1. Create a "custom-format" dump of the database you want to upgrade (let's call it `olddb`) include binary blobs (-b) and verbose (-v) output. The user can be the owner of the db, need not be postgres super account.

   ```
   pg_dump -h localhost -p 5432 -U postgres -Fc -b -v -f "/somepath/olddb.backup" olddb
   ```

2. Do a fresh install of PostGIS in a new database -- we'll refer to this database as `newdb`. Please refer to Section 3.3.2 and Section 3.3.1 for instructions on how to do this.

   The spatial_ref_sys entries found in your dump will be restored, but they will not override existing ones in spatial_ref_sys. This is to ensure that fixes in the official set will be properly propagated to restored databases. If for any reason you really want your own overrides of standard entries just don't load the spatial_ref_sys.sql file when creating the new db.

   If your database is really old or you know you've been using long deprecated functions in your views and functions, you might need to load `legacy.sql` for all your functions and views etc. to properly come back. Only do this if _really_ needed. Consider upgrading your views and functions before dumping instead, if possible. The deprecated functions can be later removed by loading `uninstall_legacy.sql`.

3. Restore your backup into your fresh `newdb` database using postgis_restore.pl. Unexpected errors, if any, will be printed to the standard error stream by psql. Keep a log of those.

   ```
   perl utils/postgis_restore.pl "/somepath/olddb.backup" | psql -h localhost -p 5432 -U ↩
       postgres newdb 2
   > errors.txt
   ```

Errors may arise in the following cases:

1. Some of your views or functions make use of deprecated PostGIS objects. In order to fix this you may try loading `legacy.sql` script prior to restore or you'll have to restore to a version of PostGIS which still contains those objects and try a migration again after porting your code. If the `legacy.sql` way works for you, don't forget to fix your code to stop using deprecated functions and drop them loading `uninstall_legacy.sql`.

2. Some custom records of spatial_ref_sys in dump file have an invalid SRID value. Valid SRID values are bigger than 0 and smaller than 999000. Values in the 999000.999999 range are reserved for internal use while values > 999999 can't be used at all. All your custom records with invalid SRIDs will be retained, with those > 999999 moved into the reserved range, but the spatial_ref_sys table would lose a check constraint guarding for that invariant to hold and possibly also its primary key (when multiple invalid SRIDS get converted to the same reserved SRID value).

   In order to fix this you should copy your custom SRS to a SRID with a valid value (maybe in the 910000..910999 range), convert all your tables to the new srid (see UpdateGeometrySRID), delete the invalid entry from spatial_ref_sys and re-construct the check(s) with:

```
ALTER TABLE spatial_ref_sys ADD CONSTRAINT spatial_ref_sys_srid_check check (srid > 0 ↩
    AND srid < 999000 );
```

```
ALTER TABLE spatial_ref_sys ADD PRIMARY KEY(srid));
```

   If you are upgrading an old database containing french IGN cartography, you will have probably SRIDs out of range and you will see, when importing your database, issues like this :

```
WARNING: SRID 310642222 converted to 999175 (in reserved zone)
```

   In this case, you can try following steps : first throw out completely the IGN from the sql which is resulting from postgis_restore.pl. So, after having run :

```
perl utils/postgis_restore.pl "/somepath/olddb.backup"
> olddb.sql
```

   run this command :

```
grep -v IGNF olddb.sql
> olddb-without-IGN.sql
```

   Create then your newdb, activate the required Postgis extensions, and insert properly the french system IGN with : this script After these operations, import your data :

```
psql -h localhost -p 5432 -U postgres -d newdb -f olddb-without-IGN.sql  2
> errors.txt
```

Chapter 4

Data Management

4.1 Carregando dados GIS (Vector)

4.1.1 OGC Geometry

The Open Geospatial Consortium (OGC) developed the *Simple Features Access* standard (SFA) to provide a model for geospatial data. It defines the fundamental spatial type of **Geometry**, along with operations which manipulate and transform geometry values to perform spatial analysis tasks. PostGIS implements the OGC Geometry model as the PostgreSQL data types geometry and geography.

Geometry is an *abstract* type. Geometry values belong to one of its *concrete* subtypes which represent various kinds and dimensions of geometric shapes. These include the **atomic** types Point, LineString, LinearRing and Polygon, and the **collection** types MultiPoint, MultiLineString, MultiPolygon and GeometryCollection. The *Simple Features Access - Part 1: Common architecture v1.2.1* adds subtypes for the structures PolyhedralSurface, Triangle and TIN.

Geometry models shapes in the 2-dimensional Cartesian plane. The PolyhedralSurface, Triangle, and TIN types can also represent shapes in 3-dimensional space. The size and location of shapes are specified by their **coordinates**. Each coordinate has a X and Y **ordinate** value determining its location in the plane. Shapes are constructed from points or line segments, with points specified by a single coordinate, and line segments by two coordinates.

Coordinates may contain optional Z and M ordinate values. The Z ordinate is often used to represent elevation. The M ordinate contains a measure value, which may represent time or distance. If Z or M values are present in a geometry value, they must be defined for each point in the geometry. If a geometry has Z or M ordinates the **coordinate dimension** is 3D; if it has both Z and M the coordinate dimension is 4D.

Geometry values are associated with a **spatial reference system** indicating the coordinate system in which it is embedded. The spatial reference system is identified by the geometry SRID number. The units of the X and Y axes are determined by the spatial reference system. In **planar** reference systems the X and Y coordinates typically represent easting and northing, while in **geodetic** systems they represent longitude and latitude. SRID 0 represents an infinite Cartesian plane with no units assigned to its axes. See Section 4.5.

The geometry **dimension** is a property of geometry types. Point types have dimension 0, linear types have dimension 1, and polygonal types have dimension 2. Collections have the dimension of the maximum element dimension.

A geometry value may be **empty**. Empty values contain no vertices (for atomic geometry types) or no elements (for collections).

An important property of geometry values is their spatial **extent** or **bounding box**, which the OGC model calls **envelope**. This is the 2 or 3-dimensional box which encloses the coordinates of a geometry. It is an efficient way to represent a geometry's extent in coordinate space and to check whether two geometries interact.

The geometry model allows evaluating topological spatial relationships as described in Section 5.1.1. To support this the concepts of **interior**, **boundary** and **exterior** are defined for each geometry type. Geometries are topologically closed, so they always contain their boundary. The boundary is a geometry of dimension one less than that of the geometry itself.

The OGC geometry model defines validity rules for each geometry type. These rules ensure that geometry values represents realistic situations (e.g. it is possible to specify a polygon with a hole lying outside the shell, but this makes no sense geometrically and is thus invalid). PostGIS also allows storing and manipulating invalid geometry values. This allows detecting and fixing them if needed. See Section 4.4

4.1.1.1 Point

A Point is a 0-dimensional geometry that represents a single location in coordinate space.

```
POINT (1 2)
POINT Z (1 2 3)
POINT ZM (1 2 3 4)
```

4.1.1.2 LineString

A LineString is a 1-dimensional line formed by a contiguous sequence of line segments. Each line segment is defined by two points, with the end point of one segment forming the start point of the next segment. An OGC-valid LineString has either zero or two or more points, but PostGIS also allows single-point LineStrings. LineStrings may cross themselves (self-intersect). A LineString is **closed** if the start and end points are the same. A LineString is **simple** if it does not self-intersect.

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

4.1.1.3 LinearRing

A LinearRing is a LineString which is both closed and simple. The first and last points must be equal, and the line must not self-intersect.

```
LINEARRING (0 0 0, 4 0 0, 4 4 0, 0 4 0, 0 0 0)
```

4.1.1.4 Polygon

A Polygon is a 2-dimensional planar region, delimited by an exterior boundary (the shell) and zero or more interior boundaries (holes). Each boundary is a LinearRing.

```
POLYGON ((0 0 0,4 0 0,4 4 0,0 4 0,0 0 0),(1 1 0,2 1 0,2 2 0,1 2 0,1 1 0))
```

4.1.1.5 MultiPoint

A MultiPoint is a collection of Points.

```
MULTIPOINT ( (0 0), (1 2) )
```

4.1.1.6 MultiLineString

A MultiLineString is a collection of LineStrings. A MultiLineString is closed if each of its elements is closed.

```
MULTILINESTRING ( (0 0,1 1,1 2), (2 3,3 2,5 4) )
```

4.1.1.7 MultiPolygon

A MultiPolygon is a collection of non-overlapping, non-adjacent Polygons. Polygons in the collection may touch only at a finite number of points.

```
MULTIPOLYGON (((1 5, 5 5, 5 1, 1 1, 1 5)), ((6 5, 9 1, 6 1, 6 5)))
```

4.1.1.8 GeometryCollection

A GeometryCollection is a heterogeneous (mixed) collection of geometries.

```
GEOMETRYCOLLECTION ( POINT(2 3), LINESTRING(2 3, 3 4))
```

4.1.1.9 PolyhedralSurface

A PolyhedralSurface is a contiguous collection of patches or facets which share some edges. Each patch is a planar Polygon. If the Polygon coordinates have Z ordinates then the surface is 3-dimensional.

```
POLYHEDRALSURFACE Z (
  ((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)) )
```

4.1.1.10 Triangle

A Triangle is a polygon defined by three distinct non-collinear vertices. Because a Triangle is a polygon it is specified by four coordinates, with the first and fourth being equal.

```
TRIANGLE ((0 0, 0 9, 9 0, 0 0))
```

4.1.1.11 TIN

A TIN is a collection of non-overlapping Triangles representing a Triangulated Irregular Network.

```
TIN Z ( ((0 0 0, 0 0 1, 0 1 0, 0 0 0)), ((0 0 0, 0 1 0, 1 1 0, 0 0 0)) )
```

4.1.2 SQL-MM Part 3

The *ISO/IEC 13249-3 SQL Multimedia - Spatial* standard (SQL/MM) extends the OGC SFA to define Geometry subtypes containing curves with circular arcs. The SQL/MM types support 3DM, 3DZ and 4D coordinates.

Note
Todos as comparações de pontos flutuantes dentro da implementação SQL-MM são representadas com uma tolerância específica, atualmente 1E-8.

4.1.2.1 CircularString

CircularString is the basic curve type, similar to a LineString in the linear world. A single arc segment is specified by three points: the start and end points (first and third) and some other point on the arc. To specify a closed circle the start and end points are the same and the middle point is the opposite point on the circle diameter (which is the center of the arc). In a sequence of arcs the end point of the previous arc is the start point of the next arc, just like the segments of a LineString. This means that a CircularString must have an odd number of points greater than 1.

```
CIRCULARSTRING(0 0, 1 1, 1 0)

CIRCULARSTRING(0 0, 4 0, 4 4, 0 4, 0 0)
```

4.1.2.2 CompoundCurve

Uma curva composta é uma curva única e contínua que tem segmentos curvados (circulares) e lineares. Isto significa que, além de ter componentes bem formados, o ponto final de cada componente (exceto o último) deve ser coincidente com o ponto inicial do componente seguinte.

```
COMPOUNDCURVE( CIRCULARSTRING(0 0, 1 1, 1 0),(1 0, 0 1))
```

4.1.2.3 CurvePolygon

Um POLÍGONOCURVO é como um polígono, com um anel externo e zero ou mais anéis internos. A diferença é que um anel pode obter a forma de uma string circular, linear ou composta.

Assim como o PostGIS 1.4, o PostGIS suporta curvas compostas em um polígono curvo.

```
CURVEPOLYGON(
  CIRCULARSTRING(0 0, 4 0, 4 4, 0 4, 0 0),
  (1 1, 3 3, 3 1, 1 1) )
```

Example: A CurvePolygon with the shell defined by a CompoundCurve containing a CircularString and a LineString, and a hole defined by a CircularString

```
CURVEPOLYGON(
  COMPOUNDCURVE( CIRCULARSTRING(0 0,2 0, 2 1, 2 3, 4 3),
                 (4 3, 4 5, 1 4, 0 0)),
  CIRCULARSTRING(1.7 1, 1.4 0.4, 1.6 0.4, 1.6 0.5, 1.7 1) )
```

4.1.2.4 MultiCurve

A MULTICURVA é uma coleção de curvas, que podem incluir strings lineares, circulares e compostas.

```
MULTICURVE( (0 0, 5 5), CIRCULARSTRING(4 0, 4 4, 8 4))
```

4.1.2.5 MultiSurface

Esta é uma coleção de superfícies, que podem ser polígonos (lineares) ou polígonos curvos.

```
MULTISURFACE(
  CURVEPOLYGON(
    CIRCULARSTRING( 0 0, 4 0, 4 4, 0 4, 0 0),
    (1 1, 3 3, 3 1, 1 1)),
  ((10 10, 14 12, 11 10, 10 10), (11 11, 11.5 11, 11 11.5, 11 11)))
```

4.1.3 OpenGIS WKB e WKT

A especificação OpenGIS define dois caminhos padrão de expressar objetos espaciais: o Well-Known Text (WKT) e o Well-Known Binary (WKB). Ambos incluem informação sobre o tipo do objeto e as coordenadas que os formam.

A representação bem conhecida de texto do sistema de referência espacial. Um exemplo de uma representação WKT SRS é:

- POINT(0 0)
- POINT(0 0)
- POINT(0 0)
- POINT EMPTY
- LINESTRING(0 0,1 1,1 2)
- LINESTRING
- POLYGON((0 0,4 0,4 4,0 4,0 0),(1 1, 2 1, 2 2, 1 2,1 1))
- MULTIPOINT((0 0),(1 2))
- MULTIPOINT((0 0),(1 2))
- MULTIPOINT
- MULTILINESTRING((0 0,1 1,1 2),(2 3,3 2,5 4))
- MULTIPOLYGON(((0 0,4 0,4 4,0 4,0 0),(1 1,2 1,2 2,1 2,1 1)), ((-1 -1,-1 -2,-2 -2,-2 -1,-1 -1)))
- GEOMETRYCOLLECTION(POINT(2 3),LINESTRING(2 3,3 4))
- GEOMETRYCOLLECTION

Input and output of WKT is provided by the functions ST_AsText and ST_GeomFromText:

```
text WKT = ST_AsText(geometry);
geometry = ST_GeomFromText(text WKT, SRID);
```

Por exemplo, uma declaração inserida válida para criar e inserir um objeto espacial OGC seria:

```
INSERT INTO geotable ( geom, name )
  VALUES ( ST_GeomFromText('POINT(-126.4 45.32)', 312), 'A Place');
```

Well-Known Binary (WKB) provides a portable, full-precision representation of spatial data as binary data (arrays of bytes). Examples of the WKB representations of spatial objects are:

- POINT(0 0)

  WKB: 01010000000000000000F03F000000000000F03
- LINESTRING(0 0,1 1,1 2)

  WKB: 0102000000020000000000000000000040000000000000004000000000000000224000000000000002240

Input and output of WKB is provided by the functions ST_AsBinary and ST_GeomFromWKB:

```
bytea WKB = ST_AsBinary(geometry);
geometry = ST_GeomFromWKB(bytea WKB, SRID);
```

Por exemplo, uma declaração inserida válida para criar e inserir um objeto espacial OGC seria:

```
INSERT INTO geotable ( geom, name )
  VALUES ( ST_GeomFromWKB('\x01010000000000000000f03f000000000000f03f', 312), 'A Place');
```

4.2 Geometry Data Type

PostGIS implements the OGC Simple Features model by defining a PostgreSQL data type called `geometry`. It represents all of the geometry subtypes by using an internal type code (see Tipo de geometria and ST_GeometryType). This allows modelling spatial features as rows of tables defined with a column of type `geometry`.

The `geometry` data type is *opaque*, which means that all access is done via invoking functions on geometry values. Functions allow creating geometry objects, accessing or updating all internal fields, and compute new geometry values. PostGIS supports all the functions specified in the OGC *Simple feature access - Part 2: SQL option* (SFS) specification, as well many others. See Chapter 7 for the full list of functions.

Note
PostGIS follows the SFA standard by prefixing spatial functions with "ST_". This was intended to stand for "Spatial and Temporal", but the temporal part of the standard was never developed. Instead it can be interpreted as "Spatial Type".

A especificação OpenGIS também requer que o formato do armazenamento interno dos objetos espacias incluam um identificador de sistema de referência espacial (SRID). O SRID é fundamental na criação de objetos espaciais para a inserção no banco de dados.

To make querying geometry efficient PostGIS defines various kinds of spatial indexes, and spatial operators to use them. See Section 4.9 and Section 5.2 for details.

4.2.1 OpenGIS WKB e WKT

OGC SFA specifications initially supported only 2D geometries, and the geometry SRID is not included in the input/output representations. The OGC SFA specification 1.2.1 (which aligns with the ISO 19125 standard) adds support for 3D (ZYZ) and measured (XYM and XYZM) coordinates, but still does not include the SRID value.

Because of these limitations PostGIS defined extended EWKB and EWKT formats. They provide 3D (XYZ and XYM) and 4D (XYZM) coordinate support and include SRID information. Including all geometry information allows PostGIS to use EWKB as the format of record (e.g. in DUMP files).

EWKB and EWKT are used for the "canonical forms" of PostGIS data objects. For input, the canonical form for binary data is EWKB, and for text data either EWKB or EWKT is accepted. This allows geometry values to be created by casting a text value in either HEXEWKB or EWKT to a geometry value using `::geometry`. For output, the canonical form for binary is EWKB, and for text it is HEXEWKB (hex-encoded EWKB).

For example this statement creates a geometry by casting from an EWKT text value, and outputs it using the canonical form of HEXEWKB:

```
SELECT 'SRID=4;POINT(0 0)'::geometry;
  geometry
  ----------------------------------------------------
  0101000020040000000000000000000000000000000000000000
```

PostGIS EWKT output has a few differences to OGC WKT:

- For 3DZ geometries the Z qualifier is omitted:

  POINT(0 0)

  POINT(0 0)

- For 3DM geometries the M qualifier is included:

  POINT(0 0)

  POINT(0 0)

- For 4D geometries the ZM qualifier is omitted:
  POINT(0 0)
  POINT(0 0)

EWKT avoids over-specifying dimensionality and the inconsistencies that can occur with the OGC/ISO format, such as:

- POINT(0 0)
- POINT(0 0)
- POINT(0 0)

Caution
Os formatos estendidos do PostGIS estão atualmente superset de OGC (cada WKB/WKT válido é um EWKB/EWKT válido), mas isto pode airar no futuro, especificamente se OGC sai com um novo formato conflitando com nossas extensões. Assim, você NÃO DEVE confiar neste aspecto!

A seguir, exemplos das representações de textos (WKT) dos objetos espaciais das características:

- POINT(0 0 0) -- XYZ
- SRID=32632;POINT(0 0) -- XY with SRID
- POINTM(0 0 0) -- XYM
- POINT(0 0 0 0) -- XYZM
- SRID=4326;MULTIPOINTM(0 0 0,1 2 1) -- XYM with SRID
- MULTILINESTRING((0 0 0,1 1 0,1 2 1),(2 3 1,3 2 1,5 4 1))
- POLYGON((0 0 0,4 0 0,4 4 0,0 4 0,0 0 0),(1 1 0,2 1 0,2 2 0,1 2 0,1 1 0))
- MULTIPOLYGON(((0 0 0,4 0 0,4 4 0,0 4 0,0 0 0),(1 1 0,2 1 0,2 2 0,1 2 0,1 1 0)),((-1 -1 0,-1 -2 0,-2 -2 0,-2 -1 0,-1 -1 0)))
- GEOMETRYCOLLECTIONM(POINTM(2 3 9), LINESTRINGM(2 3 4, 3 4 5))
- MULTICURVE((0 0, 5 5), CIRCULARSTRING(4 0, 4 4, 8 4))
- 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)))
- TRIANGLE ((0 0, 0 9, 9 0, 0 0))
- 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)))

Entrada/Saída destes formatos estão disponíveis usando as seguintes interfaces:

```
bytea EWKB = ST_AsEWKB(geometry);
text EWKT = ST_AsEWKT(geometry);
geometry = ST_GeomFromEWKB(bytea EWKB);
geometry = ST_GeomFromEWKT(text EWKT);
```

Por exemplo, uma declaração inserida válida para criar e inserir um objeto espacial seria:

```
INSERT INTO geotable ( geom, name )
  VALUES ( ST_GeomFromEWKT('SRID=312;POINTM(-126.4 45.32 15)'), 'A Place' )
```

4.3 Tipo de geografia POstGIS

O tipo de geografia fornece suporte natural para características representadas nas coordenadas"geográficas" (às vezes chamadas de coordenadas "geodéticas", ou "lat/lon", ou 'lon/lat"). As coordenadas geográficas são coordenadas esféricas expressadas em unidades angulares (graus).

A base para a geometria PostGIS é um plano. O menor caminho entre dois pontos no plano é uma linha. Isso quer dizer que cálculos em geometrias (áreas, distâncias, cumprimentos, interseções etc) podem ser feitos usando matemática cartesiana e vetores de linhas.

A base para a geometria PostGIS é um plano. O menor caminho entre dois pontos no plano é uma linha. Isso quer dizer que cálculos em geometrias (áreas, distâncias, cumprimentos, interseções etc) podem ser feitos usando matemática cartesiana e vetores de linhas.

Devido à matemática fundamental ser muito mais complicada, existem poucas funções definidas pela geografia em vez da geometria. Ao longo do tempo, à media que os algorítimos forem adicionados, as capacidades da geografia serão expandidas.

Like the geometry data type, geography data is associated with a spatial reference system via a spatial reference system identifier (SRID). Any geodetic (long/lat based) spatial reference system defined in the `spatial_ref_sys` table can be used. (Prior to PostGIS 2.2, the geography type supported only WGS 84 geodetic (SRID:4326)). You can add your own custom geodetic spatial reference system as described in Section 4.5.2.

For all spatial reference systems the units returned by measurement functions (e.g. ST_Distance, ST_Length, ST_Perimeter, ST_Area) and for the distance argument of ST_DWithin are in meters.

4.3.1 Criando uma Tabela Espacial

You can create a table to store geography data using the CREATE TABLE SQL statement with a column of type `geography`. The following example creates a table with a geography column storing 2D LineStrings in the WGS84 geodetic coordinate system (SRID 4326):

```
CREATE TABLE global_points (
    id SERIAL PRIMARY KEY,
    name VARCHAR(64),
    location geography(POINT,4326)
  );
```

The geography type supports two optional type modifiers:

- Os valores permitidos para o modificador de tipo são: PONTO, LINESTRING, POLÍGONO, MULTIPONTO, MULTILINESTRING, MULTIPOLÍGONO. O modificador também suporta restrições de dimensionalidade através de sufixos: Z, M, e ZM. Então, por exemplo, um modificador de 'LINESTRINGM' só permitiria line strings com três dimensões, e trataria a terceira dimensão como uma medida. Da mesma forma, 'PONTOZM' esperaria dados de quatro dimensões.
- the SRID modifier restricts the spatial reference system SRID to a particular number. If omitted, the SRID defaults to 4326 (WGS84 geodetic), and all calculations are performed using WGS84.

Examples of creating tables with geography columns:

- Create a table with 2D POINT geography with the default SRID 4326 (WGS84 long/lat):

```
CREATE TABLE ptgeogwgs(gid serial PRIMARY KEY, geog geography(POINT) );
```

- Create a table with 2D POINT geography in NAD83 longlat:

```
CREATE TABLE ptgeognad83(gid serial PRIMARY KEY, geog geography(POINT,4269) );
```

- Create a table with 3D (XYZ) POINTs and an explicit SRID of 4326:

```
CREATE TABLE ptzgeogwgs84(gid serial PRIMARY KEY, geog geography(POINTZ,4326) );
```

- Create a table with 2D LINESTRING geography with the default SRID 4326:

```
CREATE TABLE lgeog(gid serial PRIMARY KEY, geog geography(LINESTRING) );
```

- Create a table with 2D POLYGON geography with the SRID 4267 (NAD 1927 long lat):

```
CREATE TABLE lgeognad27(gid serial PRIMARY KEY, geog geography(POLYGON,4267) );
```

Geography fields are registered in the `geography_columns` system view. You can query the `geography_columns` view and see that the table is listed:

```
SELECT * FROM geography_columns;
```

Criar um índice funciona da mesma forma que uma GEOMETRIA. O PostGIS irá notar que o tipo de coluna é GEOGRAFIA e criará um índice baseado em esfera apropriado em vez do de costume usado para GEOMETRIA.

```
-- Index the test table with a spherical index
CREATE INDEX global_points_gix ON global_points USING GIST ( location );
```

4.3.2 Tipo de geografia POstGIS

You can insert data into geography tables in the same way as geometry. Geometry data will autocast to the geography type if it has SRID 4326. The EWKT and EWKB formats can also be used to specify geography values.

```
-- Add some data into the test table
INSERT INTO global_points (name, location) VALUES ('Town', 'SRID=4326;POINT(-110 30)');
INSERT INTO global_points (name, location) VALUES ('Forest', 'SRID=4326;POINT(-109 29)');
INSERT INTO global_points (name, location) VALUES ('London', 'SRID=4326;POINT(0 49)');
```

Any geodetic (long/lat) spatial reference system listed in `spatial_ref_sys` table may be specified as a geography SRID. Non-geodetic coordinate systems raise an error if used.

```
-- NAD 83 lon/lat
SELECT 'SRID=4269;POINT(-123 34)'::geography;
                    geography
----------------------------------------------------
 0101000020AD10000000000000C05EC00000000000004140
```

```
-- NAD27 lon/lat
SELECT 'SRID=4267;POINT(-123 34)'::geography;
                    geography
----------------------------------------------------
 0101000020AB10000000000000C05EC00000000000004140
```

```
-- NAD83 UTM zone meters - gives an error since it is a meter-based planar projection
SELECT 'SRID=26910;POINT(-123 34)'::geography;

ERROR:  Only lon/lat coordinate systems are supported in geography.
```

As funções de consulta e medida usam unidades em metros. Então, os parâmetros de distância deveriam ser esperados em metros (ou metros quadrados para áreas).

```
-- A distance query using a 1000km tolerance
SELECT name FROM global_points WHERE ST_DWithin(location, 'SRID=4326;POINT(-110 29)':: ↩
    geography, 1000000);
```

You can see the power of geography in action by calculating how close a plane flying a great circle route from Seattle to London (LINESTRING(-122.33 47.606, 0.0 51.5)) comes to Reykjavik (POINT(-21.96 64.15)) (map the route).

O tipo GEOGRAFIA calcula a verdadeira menor distância sobre a esfera entre Reykjavik e o grande caminho de voo circular entre Seattle e Londres.

```
-- Distance calculation using GEOGRAPHY
SELECT ST_Distance('LINESTRING(-122.33 47.606, 0.0 51.5)'::geography, 'POINT(-21.96 64.15) ↩
    '::geography);
   st_distance
-----------------
 122235.23815667
```

Great Circle mapper A GEOMETRIA calcula a distância cartesiana insignificante entre Reykjavik e o caminho direto de Seattle para Londres marcado em um mapa. As unidades nominais do resultado podem ser chamadas de "graus", mas o resultado não corresponde a nenhuma diferença angular verdadeira entre os pontos, então, chamá-las de "graus" é incoerente.

```
-- Distance calculation using GEOMETRY
SELECT ST_Distance('LINESTRING(-122.33 47.606, 0.0 51.5)'::geometry, 'POINT(-21.96 64.15) ↩
    '::geometry);
      st_distance
--------------------
 13.342271221453624
```

4.3.3 Quando usar o tipo de dados Geografia sobre os dados Geometria

The geography data type allows you to store data in longitude/latitude coordinates, but at a cost: there are fewer functions defined on GEOGRAPHY than there are on GEOMETRY; those functions that are defined take more CPU time to execute.

O tipo que você escolheu deveria ser condicionado da área de trabalho esperada da aplicação que você está construindo. Seus dados irão abranger o globo ou uma grande área continental, ou é local para um estado, condado ou município?

- Se seus dados estiverem contidos em uma pequena área, talvez perceba que escolher uma projeção apropriada e usar GEOMETRIA é a melhor solução, em termos de desempenho e funcionalidades disponíveis.
- Se seus dados são globais ou cobrem uma região continental, você pode perceber que GEOGRAFIA permite que você construa uma sistema sem ter que se preocupar com detalhes de projeção. Você armazena seus dados em longitude/latitude, e usa as funções que foram definidas em GEOGRAFIA.
- Se você não entende de projeções, não quer aprender sobre elas e está preparado para aceitar as limitações em funcionalidade disponíveis em GEOGRAFIA, então pode ser mais fácil se usar GEOGRAFIA em vez de GEOMETRIA. Simplesmente carregue seus dados como longitude/latitude e comece a partir daqui.

Recorra a Section 12.11 para uma comparação entre o que é suportado pela Geografia vs. Geometria. Para uma breve lista e descrição das funções da Geografia, recorra a Section 12.4

4.3.4 FAQ de Geografia Avançada

1. *Você calcula na esfera ou esferoide?*

   Por padrão, todos os cálculos de distância e área são feitos no esferoide. Você irá encontrar que os resultados dos cálculos nas áreas locais combinam com os resultados locais planares em boas projeções locais. Em grandes áreas, os cálculos esferoidais são mais precisos que os feitos em um plano projetado. Todas as funções de geografia têm a opção de usar um cálculo esférico, configurando um parâmetro booleano final para 'FALSO'. isto irá acelerar os cálculos, particularmente para casos onde as geometrias são bem simples.

2. *E a linha de data e os pólos?*

   Nenhum cálculo possui a compreensão de linha de data ou polos, as coordenadas são esféricas (longitude/latitude), então uma forma que cruza a linha de data não é, de um ponto de cálculo de view, diferente de nenhuma outra forma.

3. *Qual é o maior arco que pode ser processado?*

   Nós usamos grandes arcos círculos como a "linha de interpolação" entre dois pontos. Isso significa que quaisquer dois pontos estão de fato juntaram-se de duas maneiras, depende qual direção você vá no grande círculo. Todo o nosso código assume que os pontos estão juntos pelo *menor* dos dois caminhos ao longo do grande círculo. Como consequência, formas que têm arcos de mais de 180 graus não serão modeladas corretamente.

4. *Por que é tão lento para calcular a área da Europa / Rússia / insira uma grande região geográfica aqui ?*

   Porque o polígono é muito grande! Grandes áreas são ruins por duas razões: seus limites são grandes, logo o índice tende a puxar o traço, não importa qual consulta você execute; o número de vértices é enorme, e testes (distância, contenção) têm que atravessar a lista de vértices pelo meno uma vez e algumas vezes, N vezes (com N sendo o número de vértices em outra característica candidata). As with GEOMETRY, we recommend that when you have very large polygons, but are doing queries in small areas, you "denormalize" your geometric data into smaller chunks so that the index can effectively subquery parts of the object and so queries don't have to pull out the whole object every time. Please consult ST_Subdivide function documentation. Just because you *can* store all of Europe in one polygon doesn't mean you *should*.

4.4 Geometry Validation

PostGIS is compliant with the Open Geospatial Consortium's (OGC) Simple Features specification. That standard defines the concepts of geometry being *simple* and *valid*. These definitions allow the Simple Features geometry model to represent spatial objects in a consistent and unambiguous way that supports efficient computation. (Note: the OGC SF and SQL/MM have the same definitions for simple and valid.)

4.4.1 Simple Geometry

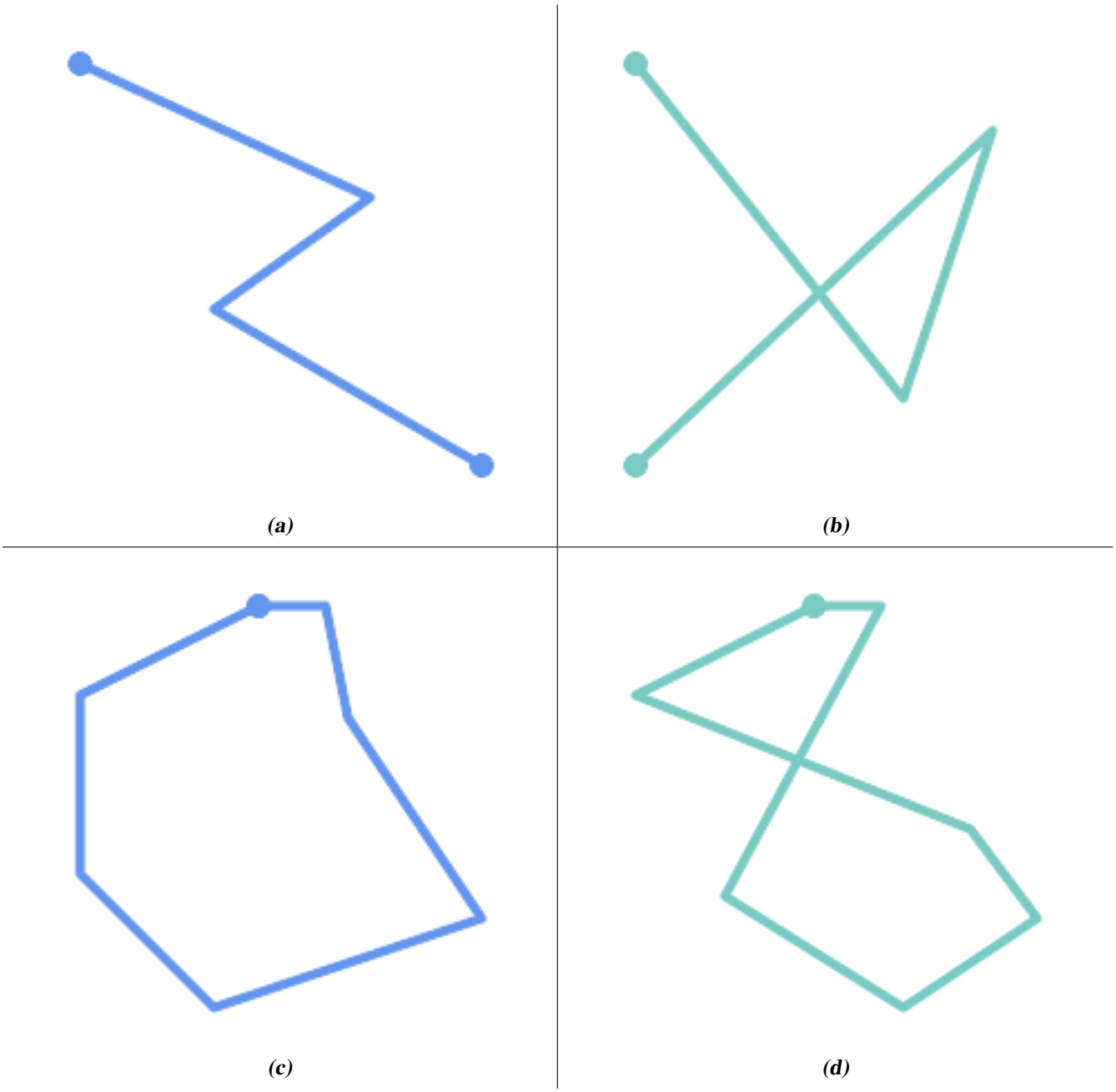
A *simple* geometry is one that has no anomalous geometric points, such as self intersection or self tangency.

Um `POINT` é herdado *simple* como um objeto geométrico 0-dimensional.

`MULTIPOINT`s são *simple* se nenhuma de duas coordenadas (`POINT`s) forem iguais (tenham o valor de coordenadas idêntico).

A `LINESTRING` is *simple* if it does not pass through the same point twice, except for the endpoints. If the endpoints of a simple LineString are identical it is called *closed* and referred to as a Linear Ring.

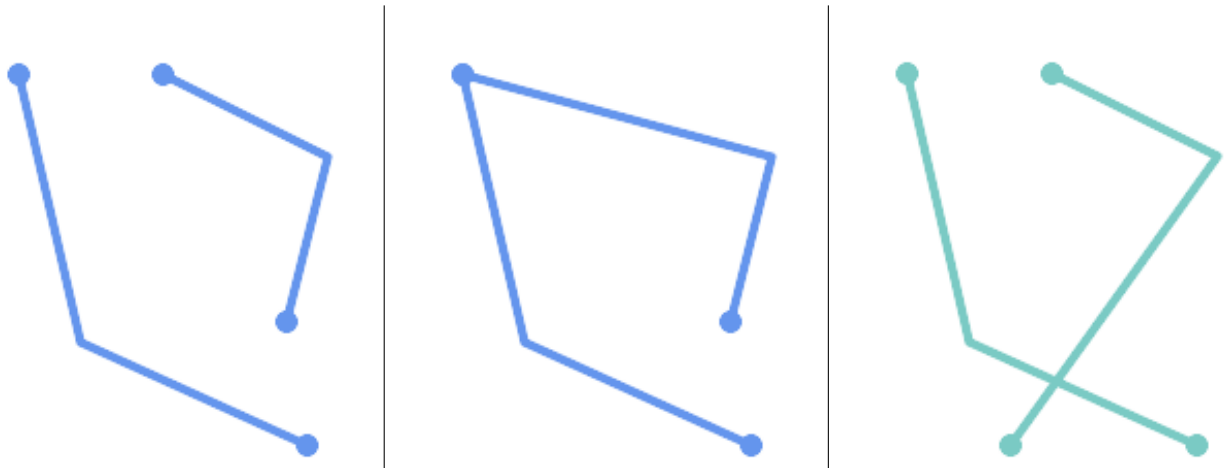
***(a)** and **(c)** are simple `LINESTRING`s. **(b)** and **(d)** are not simple. **(c)** is a closed Linear Ring.*



(a) ***(b)*** ***(c)*** ***(d)***

A `MULTILINESTRING` is *simple* only if all of its elements are simple and the only intersection between any two elements occurs at points that are on the boundaries of both elements.

(e) *and* ***(f)*** *are simple* `MULTILINESTRING`*s.* ***(g)*** *is not simple.*



(e) ***(f)*** ***(g)***

POLYGONs are formed from linear rings, so valid polygonal geometry is always *simple*.

To test if a geometry is simple use the ST_IsSimple function:

```
SELECT
   ST_IsSimple('LINESTRING(0 0, 100 100)') AS straight,
   ST_IsSimple('LINESTRING(0 0, 100 100, 100 0, 0 100)') AS crossing;

 straight | crossing
----------+----------
 t        | f
```

Generally, PostGIS functions do not require geometric arguments to be simple. Simplicity is primarily used as a basis for defining geometric validity. It is also a requirement for some kinds of spatial data models (for example, linear networks often disallow lines that cross). Multipoint and linear geometry can be made simple using ST_UnaryUnion.

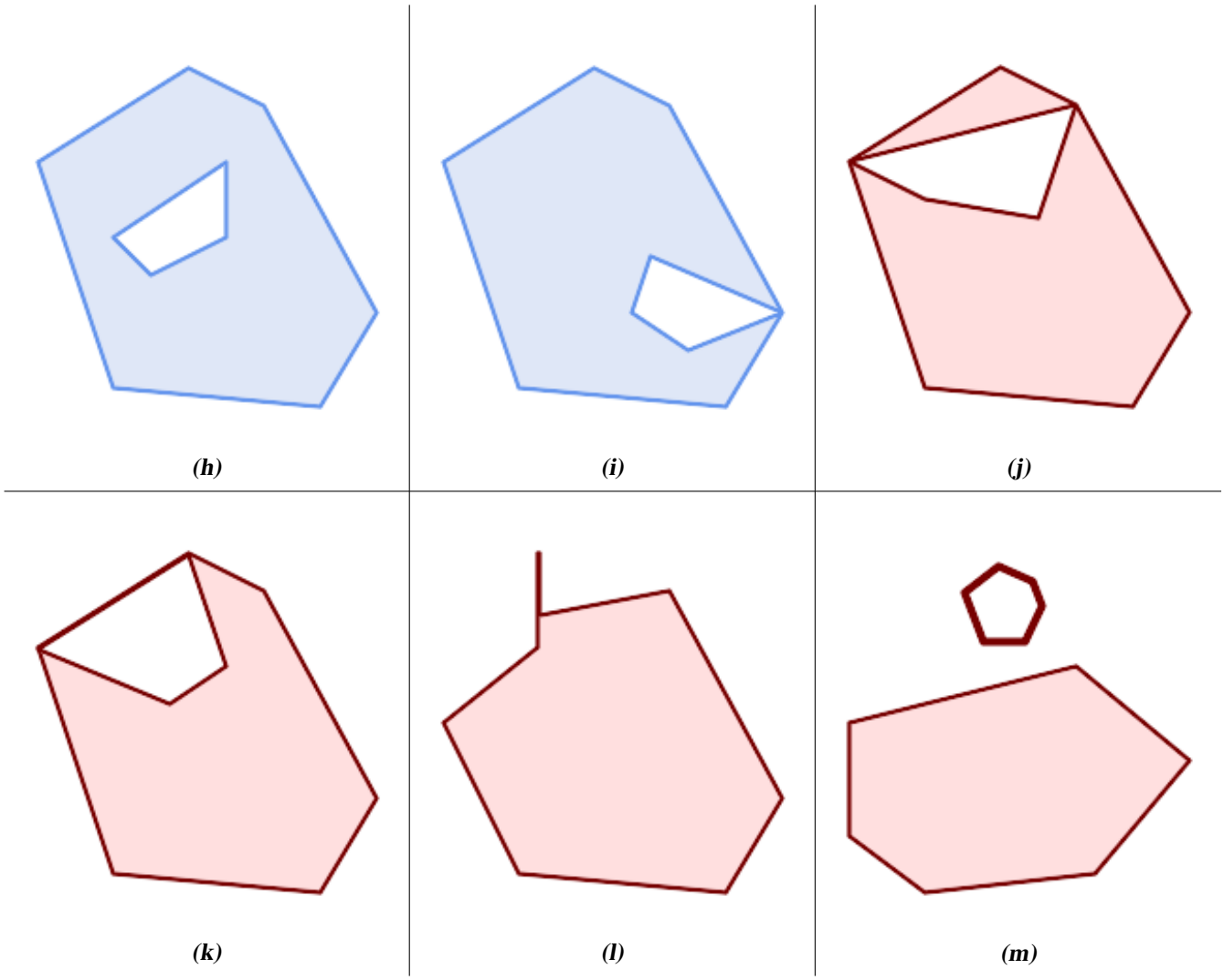
4.4.2 Valid Geometry

Geometry validity primarily applies to 2-dimensional geometries (POLYGONs and MULTIPOLYGONs) . Validity is defined by rules that allow polygonal geometry to model planar areas unambiguously.

A POLYGON is *valid* if:

1. the polygon boundary rings (the exterior shell ring and interior hole rings) are *simple* (do not cross or self-touch). Because of this a polygon cannnot have cut lines, spikes or loops. This implies that polygon holes must be represented as interior rings, rather than by the exterior ring self-touching (a so-called "inverted hole").
2. boundary rings do not cross
3. boundary rings may touch at points but only as a tangent (i.e. not in a line)
4. interior rings are contained in the exterior ring
5. the polygon interior is simply connected (i.e. the rings must not touch in a way that splits the polygon into more than one part)

(h) *and* ***(i)*** *are valid* POLYGONs. ***(j-m)*** *are invalid.* ***(j)*** *can be represented as a valid* MULTIPOLYGON.



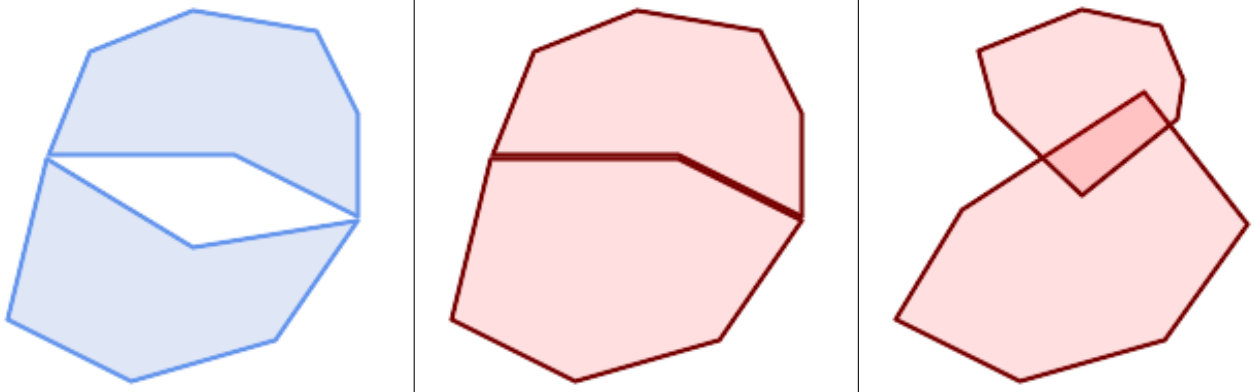
(h) ***(i)*** ***(j)***

(k) ***(l)*** ***(m)***

A `MULTIPOLYGON` is *valid* if:

1. its element `POLYGON`s are valid
2. elements do not overlap (i.e. their interiors must not intersect)
3. elements touch only at points (i.e. not along a line)

(n) *is a valid* `MULTIPOLYGON`. ***(o)*** *and* ***(p)*** *are invalid.*



(n) *(o)* *(p)*

These rules mean that valid polygonal geometry is also *simple*.

For linear geometry the only validity rule is that LINESTRINGs must have at least two points and have non-zero length (or equivalently, have at least two distinct points.) Note that non-simple (self-intersecting) lines are valid.

```
SELECT
   ST_IsValid('LINESTRING(0 0, 1 1)') AS len_nonzero,
   ST_IsValid('LINESTRING(0 0, 0 0, 0 0)') AS len_zero,
   ST_IsValid('LINESTRING(10 10, 150 150, 180 50, 20 130)') AS self_int;

 len_nonzero | len_zero | self_int
-------------+----------+----------
 t           | f        | t
```

POINT and MULTIPOINT geometries have no validity rules.

4.4.3 Managing Validity

PostGIS allows creating and storing both valid and invalid Geometry. This allows invalid geometry to be detected and flagged or fixed. There are also situations where the OGC validity rules are stricter than desired (examples of this are zero-length linestrings and polygons with inverted holes.)

Many of the functions provided by PostGIS rely on the assumption that geometry arguments are valid. For example, it does not make sense to calculate the area of a polygon that has a hole defined outside of the polygon, or to construct a polygon from a non-simple boundary line. Assuming valid geometric inputs allows functions to operate more efficiently, since they do not need to check for topological correctness. (Notable exceptions are that zero-length lines and polygons with inversions are generally handled correctly.) Also, most PostGIS functions produce valid geometry output if the inputs are valid. This allows PostGIS functions to be chained together safely.

If you encounter unexpected error messages when calling PostGIS functions (such as "GEOS Intersection() threw an error!"), you should first confirm that the function arguments are valid. If they are not, then consider using one of the techniques below to ensure the data you are processing is valid.

Note
If a function reports an error with valid inputs, then you may have found an error in either PostGIS or one of the libraries it uses, and you should report this to the PostGIS project. The same is true if a PostGIS function returns an invalid geometry for valid input.

To test if a geometry is valid use the ST_IsValid function:

```
SELECT ST_IsValid('POLYGON ((20 180, 180 180, 180 20, 20 20, 20 180))');
-----------------
 t
```

Information about the nature and location of an geometry invalidity are provided by the ST_IsValidDetail function:

```
SELECT valid, reason, ST_AsText(location) AS location
    FROM ST_IsValidDetail('POLYGON ((20 20, 120 190, 50 190, 170 50, 20 20))') AS t;

 valid |      reason       |                  location
-------+-------------------+--------------------------------------------
 f     | Self-intersection | POINT(91.51162790697674 141.56976744186045)
```

In some situations it is desirable to correct invalid geometry automatically. Use the ST_MakeValid function to do this. (`ST_MakeValid` is a case of a spatial function that *does* allow invalid input!)

By default, PostGIS does not check for validity when loading geometry, because validity testing can take a lot of CPU time for complex geometries. If you do not trust your data sources, you can enforce a validity check on your tables by adding a check constraint:

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

4.5 The SPATIAL_REF_SYS Table and Spatial Reference Systems

A Spatial Reference System (SRS) (also called a Coordinate Reference System (CRS)) defines how geometry is referenced to locations on the Earth's surface. There are three types of SRS:

- A **geodetic** SRS uses angular coordinates (longitude and latitude) which map directly to the surface of the earth.
- A **projected** SRS uses a mathematical projection transformation to "flatten" the surface of the spheroidal earth onto a plane. It assigns location coordinates in a way that allows direct measurement of quantities such as distance, area, and angle. The coordinate system is Cartesian, which means it has a defined origin point and two perpendicular axes (usually oriented North and East). Each projected SRS uses a stated length unit (usually metres or feet). A projected SRS may be limited in its area of applicability to avoid distortion and fit within the defined coordinate bounds.
- A **local** SRS is a Cartesian coordinate system which is not referenced to the earth's surface. In PostGIS this is specified by a SRID value of 0.

There are many different spatial reference systems in use. Common SRSes are standardized in the European Petroleum Survey Group EPSG database. For convenience PostGIS (and many other spatial systems) refers to SRS definitions using an integer identifier called a SRID.

A geometry is associated with a Spatial Reference System by its SRID value, which is accessed by ST_SRID. The SRID for a geometry can be assigned using ST_SetSRID. Some geometry constructor functions allow supplying a SRID (such as ST_Point and ST_MakeEnvelope). The EWKT format supports SRIDs with the `SRID=n;` prefix.

Spatial functions processing pairs of geometries (such as overlay and relationship functions) require that the input geometries are in the same spatial reference system (have the same SRID). Geometry data can be transformed into a different spatial reference system using ST_Transform and ST_TransformPipeline. Geometry returned from functions has the same SRS as the input geometries.

4.5.1 SPATIAL_REF_SYS Table

The SPATIAL_REF_SYS table used by PostGIS is an OGC-compliant database table that defines the available spatial reference systems. It holds the numeric SRIDs and textual descriptions of the coordinate systems.

A tabela SPATIAL_REF_SYS de definição está como segue:

```
CREATE TABLE spatial_ref_sys (
  srid       INTEGER NOT NULL PRIMARY KEY,
  auth_name  VARCHAR(256),
  auth_srid  INTEGER,
  srtext     VARCHAR(2048),
  proj4text  VARCHAR(2048)
)
```

As opções da commandline são:

srid Um valor inteiro que só identifica o Sistema de Referenciação Espacial (SRS) dentro do banco de dados.

auth_name O nome do corpo padrão ou corpos padrẽos que estão sendo citados por este sistema de referência. Por exemplo, "EPSG" seria um AUTH_NAME válido.

auth_srid The ID of the Spatial Reference System as defined by the Authority cited in the auth_name. In the case of EPSG, this is the EPSG code.

srtext A representação bem conhecida de texto do sistema de referência espacial. Um exemplo de uma representação WKT SRS é:

```
PROJCS["NAD83 / UTM Zone 10N",
  GEOGCS["NAD83",
        DATUM["North_American_Datum_1983",
          SPHEROID["GRS 1980",6378137,298.257222101]
        ],
        PRIMEM["Greenwich",0],
        UNIT["degree",0.0174532925199433]
  ],
  PROJECTION["Transverse_Mercator"],
  PARAMETER["latitude_of_origin",0],
  PARAMETER["central_meridian",-123],
  PARAMETER["scale_factor",0.9996],
  PARAMETER["false_easting",500000],
  PARAMETER["false_northing",0],
  UNIT["metre",1]
]
```

For a discussion of SRS WKT, see the OGC standard Well-known text representation of coordinate reference systems.

proj4text O PostGIS usa a biblioteca Proj4 para fornecer capacidades de transformação de coordenada. A coluna PROJ4TEXT contém a string de definição da coordenada Proj4 para um SRID específico. Por exemplo:

```
+proj=utm +zone=10 +ellps=clrk66 +datum=NAD27 +units=m
```

Para maiores informações a respeito, veja o website do Proj4 http://trac.osgeo.org/proj/. O arquivo spatial_ref_sys.sql contém as definições SRTEXT e PROJ4TEXT para todas as projeções EPSG.

When retrieving spatial reference system definitions for use in transformations, PostGIS uses fhe following strategy:

- If auth_name and auth_srid are present (non-NULL) use the PROJ SRS based on those entries (if one exists).
- If srtext is present create a SRS using it, if possible.
- If proj4text is present create a SRS using it, if possible.

4.5.2 The SPATIAL_REF_SYS Table and Spatial Reference Systems

The PostGIS `spatial_ref_sys` table contains over 3000 of the most common spatial reference system definitions that are handled by the PROJ projection library. But there are many coordinate systems that it does not contain. You can add SRS definitions to the table if you have the required information about the spatial reference system. Or, you can define your own custom spatial reference system if you are familiar with PROJ constructs. Keep in mind that most spatial reference systems are regional and have no meaning when used outside of the bounds they were intended for.

Uma ótima fonte para encontrar sistemas de referência espacial não definidos na configuração central é http://spatialreference.org/-

Alguns dos sistemas de referência espacial mais comumente usados são: 4326 - WGS 84 Long Lat, 4269 - NAD 83 Long Lat, 3395 - WGS 84 World Mercator, 2163 - US National Atlas Equal Area, Spatial reference systems para cadaNAD 83, WGS 84 UTM zona - zonas UTM são as mais ideais para medição, mas só cobrem 6-graus regiões.

Vários estados dos EUA no sistema de referência espacial (em metros ou pés) - normalmente um ou 2 existem por estado. A maioria dos que estão em metros estão no centro, mas muitos dos que estão em pés ou foram criados por ESRI precisarão de spatialreference.org.

You can even define non-Earth-based coordinate systems, such as Mars 2000 This Mars coordinate system is non-planar (it's in degrees spheroidal), but you can use it with the `geography` type to obtain length and proximity measurements in meters instead of degrees.

Here is an example of loading a custom coordinate system using an unassigned SRID and the PROJ definition for a US-centric Lambert Conformal projection:

```
INSERT INTO spatial_ref_sys (srid, proj4text)
VALUES ( 990000,
  '+proj=lcc  +lon_0=-95 +lat_0=25 +lat_1=25 +lat_2=25 +x_0=0 +y_0=0 +datum=WGS84 +units=m  ↩
     +no_defs'
);
```

4.6 Criando uma Tabela Espacial

4.6.1 Criando uma Tabela Espacial

You can create a table to store geometry data using the CREATE TABLE SQL statement with a column of type `geometry`. The following example creates a table with a geometry column storing 2D (XY) LineStrings in the BC-Albers coordinate system (SRID 3005):

```
CREATE TABLE roads (
    id SERIAL PRIMARY KEY,
    name VARCHAR(64),
    geom geometry(LINESTRING,3005)
  );
```

The `geometry` type supports two optional **type modifiers**:

- Os valores permitidos para o modificador de tipo são: PONTO, LINESTRING, POLÍGONO, MULTIPONTO, MULTILINESTRING, MULTIPOLÍGONO. O modificador também suporta restrições de dimensionalidade através de sufixos: Z, M, e ZM. Então, por exemplo, um modificador de 'LINESTRINGM' só permitiria line strings com três dimensões, e trataria a terceira dimensão como uma medida. Da mesma forma, 'PONTOZM' esperaria dados de quatro dimensões.
- the **SRID modifier** restricts the spatial reference system SRID to a particular number. If omitted, the SRID defaults to 0.

Examples of creating tables with geometry columns:

- Create a table holding any kind of geometry with the default SRID:

```
CREATE TABLE geoms(gid serial PRIMARY KEY, geom geometry );
```

- Create a table with 2D POINT geometry with the default SRID:

```
CREATE TABLE pts(gid serial PRIMARY KEY, geom geometry(POINT) );
```

- Create a table with 3D (XYZ) POINTs and an explicit SRID of 3005:

```
CREATE TABLE pts(gid serial PRIMARY KEY, geom geometry(POINTZ,3005) );
```

- Create a table with 4D (XYZM) LINESTRING geometry with the default SRID:

```
CREATE TABLE lines(gid serial PRIMARY KEY, geom geometry(LINESTRINGZM) );
```

- Create a table with 2D POLYGON geometry with the SRID 4267 (NAD 1927 long lat):

```
CREATE TABLE polys(gid serial PRIMARY KEY, geom geometry(POLYGON,4267) );
```

It is possible to have more than one geometry column in a table. This can be specified when the table is created, or a column can be added using the ALTER TABLE SQL statement. This example adds a column that can hold 3D LineStrings:

```
ALTER TABLE roads ADD COLUMN geom2 geometry(LINESTRINGZ,4326);
```

4.6.2 A GEOMETRY_COLUMNS VIEW

O OpenGIS "Especificação de Características Simples para SQL" define tipos padrão de objetos GIS, as funções requeridas para manipulá-los e um conjunto de tabelas de metadados. Querendo certificar-se de que os metadados permaneçam consistentes, operações como criar e remover uma coluna espacial são carregadas para fora dos procedimentos especiais definido pelo OpenGIS.

```
\d geometry_columns
```

```
View "public.geometry_columns"
      Column       |          Type          | Modifiers
-------------------+------------------------+-----------
 f_table_catalog   | character varying(256) |
 f_table_schema    | character varying(256) |
 f_table_name      | character varying(256) |
 f_geometry_column | character varying(256) |
 coord_dimension   | integer                |
 srid              | integer                |
 type              | character varying(30)  |
```

As opções da commandline são:

f_table_catalog, f_table_schema, f_table_name O nome completo da tabela de característica que contém a coluna geométrica. Note que os termos "catálogo" e "esquema" são Oracle. Não existe um análogo do "catálogo" PostgreSQL, logo a coluna é deixada em branco -- para "esquema" o nome do esquema PostgreSQL é usado (`public` é o padrão).

\d geometry_columns O nome da coluna geométrica na tabela característica.

coord_dimension A dimensão espacial (2, 3 ou 4 dimensões) da coluna.

srid A ID do sistema de referência espacial usada pela coordenada nesta coluna. É uma referência de chave estrangeira para `SPATIAL_REF_SYS`.

type O tipo do objeto espacial. Para restringir a coluna espacial a um tipo só, use um dos: PONTO, LINESTRING, POLÍGONO, MULTIPONTO, MULTILINESTRING, MULTIPOLÍGONO, GEOMETRYCOLLECTION ou versões correspondentes XYM PONTOM, LINESTRINGM, POLÍGONOM, MULTIPOINTM, MULTILINESTRINGM, MULTIPOLÍGONOM, GEOMETRYCOLLECTIONM. Para coleções heterogêneas (do tipo mistas), você pode usar "GEOMETRIA" como o tipo.

4.6.3 Registrando manualmente as colunas geométricas em geometry_columns

Two of the cases where you may need this are the case of SQL Views and bulk inserts. For bulk insert case, you can correct the registration in the geometry_columns table by constraining the column or doing an alter table. For views, you could expose using a CAST operation. Note, if your column is typmod based, the creation process would register it correctly, so no need to do anything. Also views that have no spatial function applied to the geometry will register the same as the underlying table geometry column.

```
-- Lets say you have a view created like this
CREATE VIEW public.vwmytablemercator AS
        SELECT gid, ST_Transform(geom, 3395) As geom, f_name
        FROM public.mytable;

-- For it to register correctly
-- You need to cast the geometry
--
DROP VIEW public.vwmytablemercator;
CREATE VIEW  public.vwmytablemercator AS
        SELECT gid, ST_Transform(geom, 3395)::geometry(Geometry, 3395) As geom, f_name
        FROM public.mytable;

-- If you know the geometry type for sure is a 2D POLYGON then you could do
DROP VIEW public.vwmytablemercator;
CREATE VIEW  public.vwmytablemercator AS
        SELECT gid, ST_Transform(geom,3395)::geometry(Polygon, 3395) As geom, f_name
        FROM public.mytable;
```

```
--Lets say you created a derivative table by doing a bulk insert
SELECT poi.gid, poi.geom, citybounds.city_name
INTO myschema.my_special_pois
FROM poi INNER JOIN citybounds ON ST_Intersects(citybounds.geom, poi.geom);

-- Create 2D index on new table
CREATE INDEX idx_myschema_myspecialpois_geom_gist
  ON myschema.my_special_pois USING gist(geom);

-- If your points are 3D points or 3M points,
-- then you might want to create an nd index instead of a 2D index
CREATE INDEX my_special_pois_geom_gist_nd
        ON my_special_pois USING gist(geom gist_geometry_ops_nd);

-- To manually register this new table's geometry column in geometry_columns.
-- Note it will also change the underlying structure of the table to
-- to make the column typmod based.
SELECT populate_geometry_columns('myschema.my_special_pois'::regclass);

-- If you are using PostGIS 2.0 and for whatever reason, you
-- you need the constraint based definition behavior
-- (such as case of inherited tables where all children do not have the same type and srid)
-- set optional use_typmod argument to false
SELECT populate_geometry_columns('myschema.my_special_pois'::regclass, false);
```

Although the old-constraint based method is still supported, a constraint-based geometry column used directly in a view, will not register correctly in geometry_columns, as will a typmod one. In this example we define a column using typmod and another using constraints.

```
CREATE TABLE pois_ny(gid SERIAL PRIMARY KEY, poi_name text, cat text, geom geometry(POINT ↩
    ,4326));
SELECT AddGeometryColumn('pois_ny', 'geom_2160', 2160, 'POINT', 2, false);
```

Se executarmos em psql

```
\d pois_ny;
```

Observamos que elas são definidas de maneira diferente -- uma é typmod, outra é restrição

```
Table "public.pois_ny"
  Column   |         Type          |                      Modifiers

-----------+-----------------------+-----------------------------------------------------
 gid       | integer               | not null default nextval('pois_ny_gid_seq'::regclass)
 poi_name  | text                  |
 cat       | character varying(20) |
 geom      | geometry(Point,4326)  |
 geom_2160 | geometry              |
Indexes:
    "pois_ny_pkey" PRIMARY KEY, btree (gid)
Check constraints:
    "enforce_dims_geom_2160" CHECK (st_ndims(geom_2160) = 2)
    "enforce_geotype_geom_2160" CHECK (geometrytype(geom_2160) = 'POINT'::text
        OR geom_2160 IS NULL)
    "enforce_srid_geom_2160" CHECK (st_srid(geom_2160) = 2160)
```

Nas geometry_columns, elas registram corretamente

```
SELECT f_table_name, f_geometry_column, srid, type
        FROM geometry_columns
        WHERE f_table_name = 'pois_ny';
```

```
f_table_name | f_geometry_column | srid | type
-------------+-------------------+------+-------
pois_ny      | geom              | 4326 | POINT
pois_ny      | geom_2160         | 2160 | POINT
```

Entretanto -- se se quiséssemos criar uma view como essa

```
CREATE VIEW vw_pois_ny_parks AS
SELECT *
  FROM pois_ny
  WHERE cat='park';

SELECT f_table_name, f_geometry_column, srid, type
        FROM geometry_columns
        WHERE f_table_name = 'vw_pois_ny_parks';
```

A coluna baseada em typmod registra corretamente, mas a baseada em restrições não.

```
f_table_name   | f_geometry_column | srid |   type
-----------------+-------------------+------+----------
 vw_pois_ny_parks | geom              | 4326 | POINT
 vw_pois_ny_parks | geom_2160         |    0 | GEOMETRY
```

Isto pode modificar as versões futuras do PostGIS, mas por enquanto para forçar a restrição baseada em coluna view registrar corretamente, precisamos fazer isto:

```
DROP VIEW vw_pois_ny_parks;
CREATE VIEW vw_pois_ny_parks AS
SELECT gid, poi_name, cat,
  geom,
  geom_2160::geometry(POINT,2160) As geom_2160
  FROM pois_ny
  WHERE cat = 'park';
SELECT f_table_name, f_geometry_column, srid, type
```

```
        FROM geometry_columns
        WHERE f_table_name = 'vw_pois_ny_parks';
```

```
f_table_name   | f_geometry_column | srid | type
-----------------+-------------------+------+-------
 vw_pois_ny_parks | geom              | 4326 | POINT
 vw_pois_ny_parks | geom_2160         | 2160 | POINT
```

4.7 Carregando dados GIS (Vector)

Uma vez que tenha criado uma tabela espacial, você está pronto para atualizar os dados GIS no banco de dados. No momento, existe duas formas de colocar os dados no banco de dados PostGIS/PostgreSQL: usando as declarações SQL ou usando o shape file loader/dumper.

4.7.1 Usando SQL para recuperar dados

Se você puder converter seus dados para uma representação de texto, então usar SQL formatado pode ser mais fácil de colocar seus dados no PostGIS. Como com o Oracle e outros banco de dados SQL, dados só podem ser carregados em volume canalizando um grande arquivo de texto cheio de declarações SQL "INSERT" dentro do monitor SQL.

Um arquivo de atualização de dados (`roads.sql` por exemplo) deve se parecer com:

```
BEGIN;
INSERT INTO roads (road_id, roads_geom, road_name)
  VALUES (1,'LINESTRING(191232 243118,191108 243242)','Jeff Rd');
INSERT INTO roads (road_id, roads_geom, road_name)
  VALUES (2,'LINESTRING(189141 244158,189265 244817)','Geordie Rd');
INSERT INTO roads (road_id, roads_geom, road_name)
  VALUES (3,'LINESTRING(192783 228138,192612 229814)','Paul St');
INSERT INTO roads (road_id, roads_geom, road_name)
  VALUES (4,'LINESTRING(189412 252431,189631 259122)','Graeme Ave');
INSERT INTO roads (road_id, roads_geom, road_name)
  VALUES (5,'LINESTRING(190131 224148,190871 228134)','Phil Tce');
INSERT INTO roads (road_id, roads_geom, road_name)
  VALUES (6,'LINESTRING(198231 263418,198213 268322)','Dave Cres');
COMMIT;
```

O arquivo de dados pode ser canalizado para PostgreSQL facilmente usando o "psql" SQL monitor terminal:

```
psql -d [database] -f roads.sql
```

4.7.2 shp2pgsql: Using the ESRI Shapefile Loader

O carregador de dados `shp2pgsql` converte ESRI Shape files em SQL adequado para inserção dentro de um banco de dados PostGIS/PostgreSQL, seja em formato de geometria ou geografia. O carregador possui vários modos de operação distinguidos pelas linhas de bandeiras de comando:

Juntamente com o comando carregador shp2pgsql, existe uma interface `shp2pgsql-gui` gráfica com a maioria das opções como o carregador, mas pode ser mais fácil de usar para um carregamento único non-scripted ou se você é novo no PostGIS. Pode ser configurado como um plugin do PgAdminIII.

(c|a|d|p) Essas são opções mutualmente exclusivas:

-c Cria uma tabela nova e popula do shapefile. *Este é o modo padrão.*

-a Anexa dados do shapefile dentro do banco de dados da tabela. Note que para usar esta opção para carregar vários arquivos, eles devem ter os mesmos atributos e tipos de dados.

-d Derruba a tabela do banco de dados, criando uma nova tabela com os dados do shapefile.

-p Produz somente a criação da tabela do código SQL, sem adicionar nenhum dado de fato. Isto pode ser usado se você precisar separar completamente a tabela de criação e os passos de carregamento de dados.

-? Exibir tela de ajuda.

-D Use o formato PostgreSQL "dump" para os dados de saída. Pode ser combinado com -a, -c e -d. É muito mais rápido para carregar que o formato padrão "insert" SQL. Use isto para dados muito grandes.

-s [<FROM_SRID%gt;:]<SRID> Cria e popula as tabelas de geometria com o SRID específico. Especifica, opcionalmente, que o shapefile de entrada usa o FROM_SRID dado, caso em que as geometrias serão reprojetadas para o SRID alvo. FROM_SRID não pode ser especificado com -D.

-k Mantém identificadores (coluna, esquema e atributos). Note que os atributos no shapefile estão todos em CAIXAALTA.

-i Coage todos os inteiros para 32-bit integers padrão, não cria 64-bit bigints, mesmo se a assinatura DBF parecer justificar ele.

-I Cria um índice GiST na coluna geométrica.

-m -m `a_file_name` Especifica um arquivo contendo um conjunto de mapas de nomes (longos) de colunas para nomes de colunas DBF com 10 caracteres. O conteúdo deste arquivo é uma ou mais linhas de dois nomes separados por um espaço branco e seguindo ou liderando espaço. Por exemplo:

```
COLUMNNAME DBFFIELD1
AVERYLONGCOLUMNNAME DBFFIELD2
```

-S Gera geometrias simples em vez de MULTI geometrias. Só irá ter sucesso se todas as geometrias forem de fato únicas (ex.: um MULTIPOLÍGONO com uma única shell, ou um MULTIPONTO com um único vértice).

-t <dimensionality> Força a geometria de saída a ter dimensionalidade especificada. Use as strings seguintes para indicar a dimensionalidade: 2D, 3DZ, 3DM, 4D.

Se a entrada tiver poucas dimensões especificadas, a saída terá essas dimensões cheias com zeros. Se a entrada tiver mais dimensões especificadas, as que indesejadas serão tiradas.

-w Gera o formato WKT em vez do WKB. Note que isto pode introduzir impulsos de coordenadas para perda de precisão.

-e Execute cada declaração por si mesma, sem usar uma transação. Isto permite carregar a maioria dos dados bons quando existem geometrias ruins que geram erros. Note que não pode ser usado com a bandeira -D como o formato "dump" sempre usa a transação.

-W <encoding> Especifica codificação dos dados de entrada (arquivo dbf). Quando usado, todos os atributos do dbf são convertidos da codificação especificada para UTF8. A saída SQL resultante conterá um comando `SET CLIENT_ENCODING to UTF8`, então o backend será capaz de reconverter do UTF8 para qualquer codificação que o banco de dados estiver configurado para usar internamente.

-N <policy> Políticas para lidar com geometrias NULAS (insert*,skip,abort)

-n -n Só importa arquivo DBF. Se seus dados não possuem shapefile correspondente, ele irá trocar automaticamente para este modo e carregar só o dbf. Então, só é necessário configurar esta bandeira se você tiver um shapefile completo, e se quiser os dados atributos e nenhuma geometria.

-G Use geografia em vez de geometria (requer dados long/lat) em WGS84 long lat (SRID=4326)

-T <tablespace> Especifica o espaço para a nova tabela. Os índices continuarão usando espaço padrão a menos que o parâmetro -X também seja usado. A documentação PostgreSQL tem uma boa descrição quando usa espaços personalizados.

-X <tablespace> Especifica o espaço para os novos índices da tabela. Isto se aplica ao primeiro índice chave, e o índice GIST espacial, se -I também for usado.

-c When used, this flag will prevent the generation of `ANALYZE` statements. Without the -Z flag (default behavior), the `ANALYZE` statements will be generated.

Uma seção exemplo usando o carregador para criar um arquivo de entrada e atualizando ele pode parecer com:

```
# shp2pgsql -c -D -s 4269 -i -I shaperoads.shp myschema.roadstable > roads.sql
# psql -d roadsdb -f roads.sql
```

Uma conversão e um upload podem ser feitos em apenas um passo usando encadeamento UNIX:

```
# shp2pgsql shaperoads.shp myschema.roadstable | psql -d roadsdb
```

4.8 Criando uma Tabela Espacial

Os dados podem ser extraídos do banco da dados usando o SQL ou o Shape file loader/dumper. Na seção do SQL discutiremos alguns dos operadores disponíveis para comparações e consultas em tabelas espaciais.

4.8.1 Usando SQL para recuperar dados

O mais simples significa extrair os dados do banco de dados, é usar uma consulta SQL para reduzir o número de RELATOS e COLUNAS retornados e abandonar as colunas resultantes dentro de um arquivo de texto analisável:

```
db=# SELECT road_id, ST_AsText(road_geom) AS geom, road_name FROM roads;

road_id | geom                                       | road_name
--------+--------------------------------------------+-----------
          1 | LINESTRING(191232 243118,191108 243242) | Jeff Rd
          2 | LINESTRING(189141 244158,189265 244817) | Geordie Rd
          3 | LINESTRING(192783 228138,192612 229814) | Paul St
          4 | LINESTRING(189412 252431,189631 259122) | Graeme Ave
          5 | LINESTRING(190131 224148,190871 228134) | Phil Tce
          6 | LINESTRING(198231 263418,198213 268322) | Dave Cres
          7 | LINESTRING(218421 284121,224123 241231) | Chris Way
(6 rows)
```

Entretanto, às vezes algum tipo de restrição será necessária para cortar o número de campos retornados. No caso de restrições baseadas em atributos, só use a mesma sintaxe SQL como normal com uma tabela não espacial. No caso de restrições espaciais, os operadores seguintes são úteis/disponíveis:

ST_Intersects This function tells whether two geometries share any space.

= Isto testa se duas geometrias são geometricamente iguais. Por exemplo, se 'POLYGON((0 0,1 1,1 0,0 0))' é o mesmo que 'POLYGON((0 0,1 1,1 0,0 0))' (é).

Next, you can use these operators in queries. Note that when specifying geometries and boxes on the SQL command line, you must explicitly turn the string representations into geometries function. The 312 is a fictitious spatial reference system that matches our data. So, for example:

```
SELECT road_id, road_name
  FROM roads
  WHERE roads_geom='SRID=312;LINESTRING(191232 243118,191108 243242)'::geometry;
```

A consulta acima retornaria um único relato da tabela "ROADS_GEOM"na qual a geometria era igual ao valor.

To check whether some of the roads passes in the area defined by a polygon:

```
SELECT road_id, road_name
FROM roads
WHERE ST_Intersects(roads_geom, 'SRID=312;POLYGON((...))');
```

The most common spatial query will probably be a "frame-based" query, used by client software, like data browsers and web mappers, to grab a "map frame" worth of data for display.

Usando o operador "&&" , você pode especificar uma CAIXA3D como uma caracetrística de comparação ou uma GEOMETRIA. Entretanto, quando você especifica uma GEOMETRIA, a caixa delimitadora dela será usada para a comparação.

Using a "BOX3D" object for the frame, such a query looks like this:

```
SELECT ST_AsText(roads_geom) AS geom
FROM roads
WHERE
  roads_geom && ST_MakeEnvelope(191232, 243117,191232, 243119,312);
```

Observe o uso do SRID 312, para especificar a projeção do envelope.

4.8.2 Usando o Dumper

A tabela dumper `pgsql2shp` conecta diretamente ao banco de dados e converte uma tabela (possivelmente definida por uma consulta) em um shapefile. A sintaxe básica é:

```
pgsql2shp [<options>] <database> [<schema>.]<table>
```

```
pgsql2shp [<options>] <database> <query>
```

As opções da commandline são:

-f <filename> Atribui a saída a um filename específico.

-h <host> O hospedeiro do banco de dados para se conectar.

-p <port> A porta para conectar no hospedeiro do banco de dados.

-P <password> A senha para usar quando conectar ao banco de dados.

-u <user> O nome de usuário para usar quando conectado ao banco de dados.

-g <geometry column> No caso de tabelas com várias colunas geométricas, a coluna para usar quando atribuindo o shapefile.

-b Use um cursor binário. Isto tornará a operação mais rápida, mas não funcionará se qualquer atributo NÃO-geométrico na tabela necessitar de um cast para o texto.

-r Modo cru. Não derruba o campo `gid`, ou escapa o nome das colunas.

-m `filename` Remapeia os identificadores para nomes com dez caracteres. O conteúdo do arquivo é linhas de dois símbolos separados por um único espaço branco e nenhum espaço seguindo ou à frente: VERYLONGSYMBOL SHORTONE ANOTHERVERYLONGSYMBOL SHORTER etc.

4.9 Construindo índidces

Spatial indexes make using a spatial database for large data sets possible. Without indexing, a search for features requires a sequential scan of every record in the database. Indexing speeds up searching by organizing the data into a structure which can be quickly traversed to find matching records.

The B-tree index method commonly used for attribute data is not very useful for spatial data, since it only supports storing and querying data in a single dimension. Data such as geometry (which has 2 or more dimensions) requires an index method that supports range query across all the data dimensions. One of the key advantages of PostgreSQL for spatial data handling is that it offers several kinds of index methods which work well for multi-dimensional data: GiST, BRIN and SP-GiST indexes.

- GiST (Generalized Search Trees) dissolvem dados em "coisas de um lado", "coisas que sobrepõem", "coisas que estão dentro" e pode ser usado em vários tipos de dados, incluindo dados GIS. O PostGIS usa o índice R-Tree implementado no topo do GiST para classificar dados GIS.
- **BRIN (Block Range Index)** indexes operate by summarizing the spatial extent of ranges of table records. Search is done via a scan of the ranges. BRIN is only appropriate for use for some kinds of data (spatially sorted, with infrequent or no update). But it provides much faster index create time, and much smaller index size.
- **SP-GiST (Space-Partitioned Generalized Search Tree)** is a generic index method that supports partitioned search trees such as quad-trees, k-d trees, and radix trees (tries).

Spatial indexes store only the bounding box of geometries. Spatial queries use the index as a **primary filter** to quickly determine a set of geometries potentially matching the query condition. Most spatial queries require a **secondary filter** that uses a spatial predicate function to test a more specific spatial condition. For more information on queying with spatial predicates see Section 5.2.

See also the PostGIS Workshop section on spatial indexes, and the PostgreSQL manual.

4.9.1 Índices GiST

GiST significa "Árvores de Pesquisa Generalizada" e é uma forma genérica de classificar. Além disso, ele é usado para acelerar pesquisas em todos os tipos de estruturas de dados irregulares (arranjos inteiros, dados espectrais etc) que não são agradáveis à classificação normal B-Tree.

Uma vez que uma tabela de dados GIS excede pouco mais de mil filas, você irá querer construir um índice para acelerar pesquisas espaciais dos dados (a menos que suas pesquisas sejam baseadas em atributos, você vai querer construir um índice normal nos campos de atributo).

A sintaxe para construir um índice GiST em uma coluna "geométrica" é a seguinte:

```
CREATE INDEX [indexname] ON [tablename] USING GIST ( [geometryfield] );
```

The above syntax will always build a 2D-index. To get the an n-dimensional index for the geometry type, you can create one using this syntax:

```
CREATE INDEX [indexname] ON [tablename] USING GIST ([geometryfield] gist_geometry_ops_nd);
```

Building a spatial index is a computationally intensive exercise. It also blocks write access to your table for the time it creates, so on a production system you may want to do in in a slower CONCURRENTLY-aware way:

```
CREATE INDEX CONCURRENTLY [indexname] ON [tablename] USING GIST ( [geometryfield] );
```

After building an index, it is sometimes helpful to force PostgreSQL to collect table statistics, which are used to optimize query plans:

```
VACUUM ANALYZE [table_name] [(column_name)];
```

4.9.2 BRIN Indexes

BRIN stands for "Block Range Index". It is a general-purpose index method introduced in PostgreSQL 9.5. BRIN is a *lossy* index method, meaning that a secondary check is required to confirm that a record matches a given search condition (which is the case for all provided spatial indexes). It provides much faster index creation and much smaller index size, with reasonable read performance. Its primary purpose is to support indexing very large tables on columns which have a correlation with their physical location within the table. In addition to spatial indexing, BRIN can speed up searches on various kinds of attribute data structures (integer, arrays etc). For more information see the PostgreSQL manual.

Once a spatial table exceeds a few thousand rows, you will want to build an index to speed up spatial searches of the data. GiST indexes are very performant as long as their size doesn't exceed the amount of RAM available for the database, and as long as

you can afford the index storage size, and the cost of index update on write. Otherwise, for very large tables BRIN index can be considered as an alternative.

A BRIN index stores the bounding box enclosing all the geometries contained in the rows in a contiguous set of table blocks, called a *block range*. When executing a query using the index the block ranges are scanned to find the ones that intersect the query extent. This is efficient only if the data is physically ordered so that the bounding boxes for block ranges have minimal overlap (and ideally are mutually exclusive). The resulting index is very small in size, but is typically less performant for read than a GiST index over the same data.

Building a BRIN index is much less CPU-intensive than building a GiST index. It's common to find that a BRIN index is ten times faster to build than a GiST index over the same data. And because a BRIN index stores only one bounding box for each range of table blocks, it's common to use up to a thousand times less disk space than a GiST index.

You can choose the number of blocks to summarize in a range. If you decrease this number, the index will be bigger but will probably provide better performance.

For BRIN to be effective, the table data should be stored in a physical order which minimizes the amount of block extent overlap. It may be that the data is already sorted appropriately (for instance, if it is loaded from another dataset that is already sorted in spatial order). Otherwise, this can be accomplished by sorting the data by a one-dimensional spatial key. One way to do this is to create a new table sorted by the geometry values (which in recent PostGIS versions uses an efficient Hilbert curve ordering):

```
CREATE TABLE table_sorted AS
   SELECT * FROM table  ORDER BY geom;
```

Alternatively, data can be sorted in-place by using a GeoHash as a (temporary) index, and clustering on that index:

```
CREATE INDEX idx_temp_geohash ON table
    USING btree (ST_GeoHash( ST_Transform( geom, 4326 ), 20));
CLUSTER table USING idx_temp_geohash;
```

A sintaxe para construir um índice GiST em uma coluna "geométrica" é a seguinte:

```
CREATE INDEX [indexname] ON [tablename] USING BRIN ( [geome_col] );
```

The above syntax builds a 2D index. To build a 3D-dimensional index, use this syntax:

```
CREATE INDEX [indexname] ON [tablename]
    USING BRIN ([geome_col] brin_geometry_inclusion_ops_3d);
```

You can also get a 4D-dimensional index using the 4D operator class:

```
CREATE INDEX [indexname] ON [tablename]
    USING BRIN ([geome_col] brin_geometry_inclusion_ops_4d);
```

The above commands use the default number of blocks in a range, which is 128. To specify the number of blocks to summarise in a range, use this syntax

```
CREATE INDEX [indexname] ON [tablename]
    USING BRIN ( [geome_col] ) WITH (pages_per_range = [number]);
```

Keep in mind that a BRIN index only stores one index entry for a large number of rows. If your table stores geometries with a mixed number of dimensions, it's likely that the resulting index will have poor performance. You can avoid this performance penalty by choosing the operator class with the least number of dimensions of the stored geometries

The `geography` datatype is supported for BRIN indexing. The syntax for building a BRIN index on a geography column is:

```
CREATE INDEX [indexname] ON [tablename] USING BRIN ( [geog_col] );
```

The above syntax builds a 2D-index for geospatial objects on the spheroid.

Currently, only "inclusion support" is provided, meaning that just the `&&`, `~` and `@` operators can be used for the 2D cases (for both `geometry` and `geography`), and just the `&&&` operator for 3D geometries. There is currently no support for kNN searches.

An important difference between BRIN and other index types is that the database does not maintain the index dynamically. Changes to spatial data in the table are simply appended to the end of the index. This will cause index search performance to degrade over time. The index can be updated by performing a `VACUUM`, or by using a special function `brin_summarize_new_values`. For this reason BRIN may be most appropriate for use with data that is read-only, or only rarely changing. For more information refer to the manual.

To summarize using BRIN for spatial data:

- Index build time is very fast, and index size is very small.
- Index query time is slower than GiST, but can still be very acceptable.
- Requires table data to be sorted in a spatial ordering.
- Requires manual index maintenance.
- Most appropriate for very large tables, with low or no overlap (e.g. points), which are static or change infrequently.
- More effective for queries which return relatively large numbers of data records.

4.9.3 SP-GiST Indexes

SP-GiST stands for "Space-Partitioned Generalized Search Tree" and is a generic form of indexing for multi-dimensional data types that supports partitioned search trees, such as quad-trees, k-d trees, and radix trees (tries). The common feature of these data structures is that they repeatedly divide the search space into partitions that need not be of equal size. In addition to spatial indexing, SP-GiST is used to speed up searches on many kinds of data, such as phone routing, ip routing, substring search, etc. For more information see the PostgreSQL manual.

As it is the case for GiST indexes, SP-GiST indexes are lossy, in the sense that they store the bounding box enclosing spatial objects. SP-GiST indexes can be considered as an alternative to GiST indexes.

Once a GIS data table exceeds a few thousand rows, an SP-GiST index may be used to speed up spatial searches of the data. The syntax for building an SP-GiST index on a "geometry" column is as follows:

```
CREATE INDEX [indexname] ON [tablename] USING SPGIST ( [geometryfield] );
```

The above syntax will build a 2-dimensional index. A 3-dimensional index for the geometry type can be created using the 3D operator class:

```
CREATE INDEX [indexname] ON [tablename] USING SPGIST ([geometryfield]  ↩
    spgist_geometry_ops_3d);
```

Building a spatial index is a computationally intensive operation. It also blocks write access to your table for the time it creates, so on a production system you may want to do in in a slower CONCURRENTLY-aware way:

```
CREATE INDEX CONCURRENTLY [indexname] ON [tablename] USING SPGIST ( [geometryfield] );
```

After building an index, it is sometimes helpful to force PostgreSQL to collect table statistics, which are used to optimize query plans:

```
VACUUM ANALYZE [table_name] [(column_name)];
```

An SP-GiST index can accelerate queries involving the following operators:

- <<, &<, &>, >>, <<|, &<|, |&>, |>>, &&, @>, <@, and ~=, for 2-dimensional indexes,
- &/&, ~==, @>>, and <<@, for 3-dimensional indexes.

There is no support for kNN searches at the moment.

4.9.4 Construindo índidces

Ordinarily, indexes invisibly speed up data access: once an index is built, the PostgreSQL query planner automatically decides when to use it to improve query performance. But there are some situations where the planner does not choose to use existing indexes, so queries end up using slow sequential scans instead of a spatial index.

Se você achar que seus índices não estão sendo usados (ou seus atributos) há algumas coisas que pode fazer:

- Examine the query plan and check your query actually computes the thing you need. An erroneous JOIN, either forgotten or to the wrong table, can unexpectedly retrieve table records multiple times. To get the query plan, execute with `EXPLAIN` in front of the query.
- Make sure statistics are gathered about the number and distributions of values in a table, to provide the query planner with better information to make decisions around index usage. **VACUUM ANALYZE** will compute both.

  You should regularly vacuum your databases anyways. Many PostgreSQL DBAs run **VACUUM** as an off-peak cron job on a regular basis.
- If vacuuming does not help, you can temporarily force the planner to use the index information by using the command **SET ENABLE_SEQSCAN TO OFF;**. This way you can check whether the planner is at all able to generate an index-accelerated query plan for your query. You should only use this command for debugging; generally speaking, the planner knows better than you do about when to use indexes. Once you have run your query, do not forget to run **SET ENABLE_SEQSCAN TO ON;** so that the planner will operate normally for other queries.
- If **SET ENABLE_SEQSCAN TO OFF;** helps your query to run faster, your Postgres is likely not tuned for your hardware. If you find the planner wrong about the cost of sequential versus index scans try reducing the value of `RANDOM_PAGE_COST` in `postgresql.conf`, or use **SET RANDOM_PAGE_COST TO 1.1;**. The default value for `RANDOM_PAGE_COST` is 4.0. Try setting it to 1.1 (for SSD) or 2.0 (for fast magnetic disks). Decreasing the value makes the planner more likely to use index scans.
- If **SET ENABLE_SEQSCAN TO OFF;** does not help your query, the query may be using a SQL construct that the Postgres planner is not yet able to optimize. It may be possible to rewrite the query in a way that the planner is able to handle. For example, a subquery with an inline SELECT may not produce an efficient plan, but could possibly be rewritten using a LATERAL JOIN.

For more information see the Postgres manual section on Query Planning.

Chapter 5

Spatial Queries

The *raison d'etre* of spatial databases is to perform queries inside the database which would ordinarily require desktop GIS functionality. Using PostGIS effectively requires knowing what spatial functions are available, how to use them in queries, and ensuring that appropriate indexes are in place to provide good performance.

5.1 Determining Spatial Relationships

Spatial relationships indicate how two geometries interact with one another. They are a fundamental capability for querying geometry.

5.1.1 Dimensionally Extended 9-Intersection Model

According to the OpenGIS Simple Features Implementation Specification for SQL, "the basic approach to comparing two geometries is to make pair-wise tests of the intersections between the Interiors, Boundaries and Exteriors of the two geometries and to classify the relationship between the two geometries based on the entries in the resulting 'intersection' matrix."

In the theory of point-set topology, the points in a geometry embedded in 2-dimensional space are categorized into three sets:

Boundary

The boundary of a geometry is the set of geometries of the next lower dimension. For `POINT`s, which have a dimension of 0, the boundary is the empty set. The boundary of a `LINESTRING` is the two endpoints. For `POLYGON`s, the boundary is the linework of the exterior and interior rings.

Interior

The interior of a geometry are those points of a geometry that are not in the boundary. For `POINT`s, the interior is the point itself. The interior of a `LINESTRING` is the set of points between the endpoints. For `POLYGON`s, the interior is the areal surface inside the polygon.

Exterior

The exterior of a geometry is the rest of the space in which the geometry is embedded; in other words, all points not in the interior or on the boundary of the geometry. It is a 2-dimensional non-closed surface.

The Dimensionally Extended 9-Intersection Model (DE-9IM) describes the spatial relationship between two geometries by specifying the dimensions of the 9 intersections between the above sets for each geometry. The intersection dimensions can be formally represented in a 3x3 **intersection matrix**.

For a geometry *g* the *Interior*, *Boundary*, and *Exterior* are denoted using the notation *I(g)*, *B(g)*, and *E(g)*. Also, *dim(s)* denotes the dimension of a set *s* with the domain of `{0,1,2,F}`:

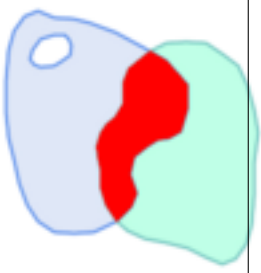
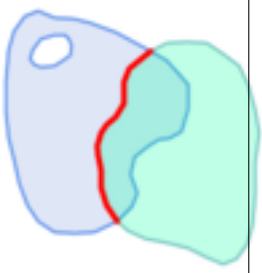
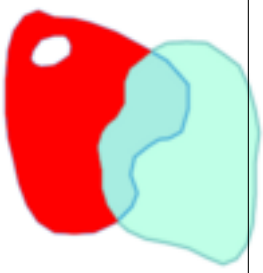
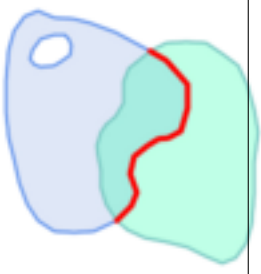
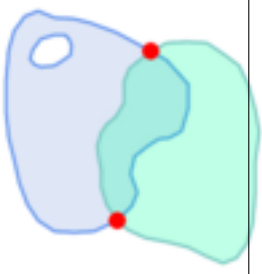
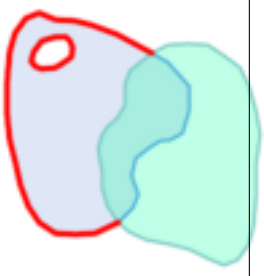
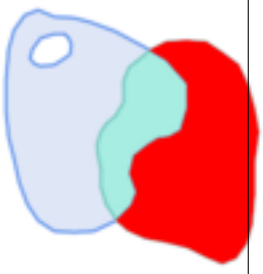
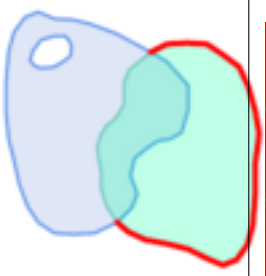
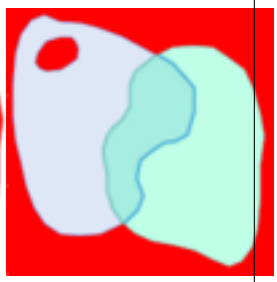
- `0` => point

- `1` => line
- `2` => area
- `F` => empty set

Using this notation, the intersection matrix for two geometries *a* and *b* is:

	Interior	**Boundary**	**Exterior**
Interior	*dim(I(a) ∩ I(b))*	*dim(I(a) ∩ B(b))*	*dim(I(a) ∩ E(b))*
Boundary	*dim(B(a) ∩ I(b))*	*dim(B(a) ∩ B(b))*	*dim(B(a) ∩ E(b))*
Exterior	*dim(E(a) ∩ I(b))*	*dim(E(a) ∩ B(b))*	*dim(E(a) ∩ E(b))*

Visually, for two overlapping polygonal geometries, this looks like:

	Interior	Boundary	Exterior
Interior	 *dim(I(a) ∩ I(b))* = **2**	 *dim(I(a) ∩ B(b)* = **1**	 *dim(I(a) ∩ E(b))* = **2**
Boundary	 *dim(B(a) ∩ I(b))* = **1**	 *dim(B(a) ∩ B(b))* = **0**	 *dim(B(a) ∩ E(b))* = **1**
Exterior	 *dim(E(a) ∩ I(b))* = **2**	 *dim(E(a) ∩ B(b))* = **1**	 *dim(E(a) ∩ E(b)* = **2**

Reading from left to right and top to bottom, the intersection matrix is represented as the text string '**212101212**'.

For more information, refer to:

- OpenGIS Simple Features Implementation Specification for SQL (version 1.1, section 2.1.13.2)
- Wikipedia: Dimensionally Extended Nine-Intersection Model (DE-9IM)
- GeoTools: Point Set Theory and the DE-9IM Matrix

5.1.2 Named Spatial Relationships

To make it easy to determine common spatial relationships, the OGC SFS defines a set of *named spatial relationship predicates*. PostGIS provides these as the functions ST_Contains, ST_Crosses, ST_Disjoint, ST_Equals, ST_Intersects, ST_Overlaps, ST_Touches, ST_Within. It also defines the non-standard relationship predicates ST_Covers, ST_CoveredBy, and ST_ContainsProperly.

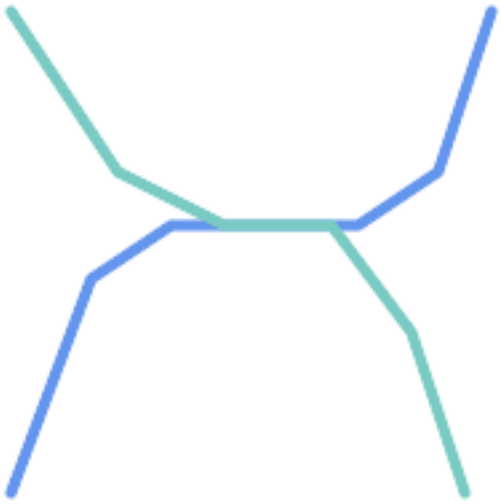
Spatial predicates are usually used as conditions in SQL `WHERE` or `JOIN` clauses. The named spatial predicates automatically use a spatial index if one is available, so there is no need to use the bounding box operator `&&` as well. For example:

```
SELECT city.name, state.name, city.geom
FROM city JOIN state ON ST_Intersects(city.geom, state.geom);
```

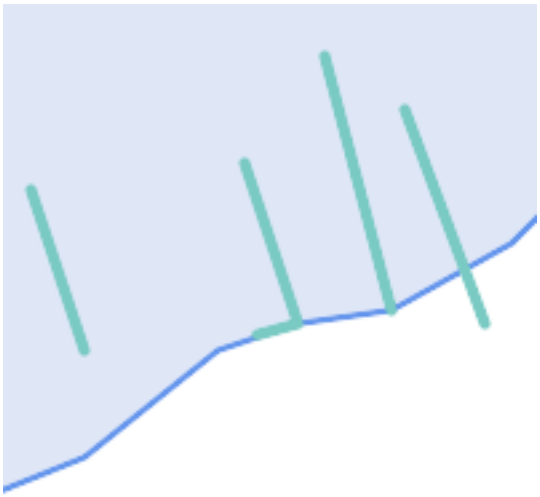
For more details and illustrations, see the PostGIS Workshop.

5.1.3 General Spatial Relationships

In some cases the named spatial relationships are insufficient to provide a desired spatial filter condition.



For example, consider a linear dataset representing a road network. It may be required to identify all road segments that cross each other, not at a point, but in a line (perhaps to validate some business rule). In this case ST_Crosses does not provide the necessary spatial filter, since for linear features it returns `true` only where they cross at a point.
A two-step solution would be to first compute the actual intersection (ST_Intersection) of pairs of road lines that spatially intersect (ST_Intersects), and then check if the intersection's ST_GeometryType is '`LINESTRING`' (properly dealing with cases that return `GEOMETRYCOLLECTION`s of `[MULTI]POINT`s, `[MULTI]LINESTRING`s, etc.).
Clearly, a simpler and faster solution is desirable.



A second example is locating wharves that intersect a lake's boundary on a line and where one end of the wharf is up on shore. In other words, where a wharf is within but not completely contained by a lake, intersects the boundary of a lake on a line, and where exactly one of the wharf's endpoints is within or on the boundary of the lake. It is possible to use a combination of spatial predicates to find the required features:

- ST_Contains(lake, wharf) = TRUE
- ST_ContainsProperly(lake, wharf) = FALSE
- ST_GeometryType(ST_Intersection(wharf, lake)) = 'LINESTRING'
- ST_NumGeometries(ST_Multi(ST_Intersection(ST_Boundary(wharf), ST_Boundary(lake)))) = 1

  ... but needless to say, this is quite complicated.

These requirements can be met by computing the full DE-9IM intersection matrix. PostGIS provides the ST_Relate function to do this:

```
SELECT ST_Relate( 'LINESTRING (1 1, 5 5)',
                  'POLYGON ((3 3, 3 7, 7 7, 7 3, 3 3))' );
st_relate
-----------
1010F0212
```

To test a particular spatial relationship, an **intersection matrix pattern** is used. This is the matrix representation augmented with the additional symbols `{T,*}`:

- `T` => intersection dimension is non-empty; i.e. is in `{0,1,2}`
- `*` => don't care

Using intersection matrix patterns, specific spatial relationships can be evaluated in a more succinct way. The ST_Relate and the ST_RelateMatch functions can be used to test intersection matrix patterns. For the first example above, the intersection matrix pattern specifying two lines intersecting in a line is '**1*1***1****':

```
-- Find road segments that intersect in a line
SELECT a.id
FROM roads a, roads b
WHERE a.id != b.id
      AND a.geom && b.geom
      AND ST_Relate(a.geom, b.geom, '1*1***1**');
```

For the second example, the intersection matrix pattern specifying a line partly inside and partly outside a polygon is '**102101FF2**':

```
-- Find wharves partly on a lake's shoreline
SELECT a.lake_id, b.wharf_id
FROM lakes a, wharfs b
WHERE a.geom && b.geom
      AND ST_Relate(a.geom, b.geom, '102101FF2');
```

5.2 Using Spatial Indexes

When constructing queries using spatial conditions, for best performance it is important to ensure that a spatial index is used, if one exists (see Section 4.9). To do this, a spatial operator or index-aware function must be used in a `WHERE` or `ON` clause of the query.

Spatial operators include the bounding box operators (of which the most commonly used is &&; see Section 7.10.1 for the full list) and the distance operators used in nearest-neighbor queries (the most common being <->; see Section 7.10.2 for the full list.)

Index-aware functions automatically add a bounding box operator to the spatial condition. Index-aware functions include the named spatial relationship predicates ST_Contains, ST_ContainsProperly, ST_CoveredBy, ST_Covers, ST_Crosses, ST_Intersects, ST_Overlaps, ST_Touches, ST_Within, ST_Within, and ST_3DIntersects, and the distance predicates ST_DWithin, ST_DFullyWithin, ST_3DDFullyWithin, and ST_3DDWithin .)

Functions such as ST_Distance do *not* use indexes to optimize their operation. For example, the following query would be quite slow on a large table:

```
SELECT geom
FROM geom_table
WHERE ST_Distance( geom, 'SRID=312;POINT(100000 200000)' ) < 100
```

This query selects all the geometries in `geom_table` which are within 100 units of the point (100000, 200000). It will be slow because it is calculating the distance between each point in the table and the specified point, ie. one `ST_Distance()` calculation is computed for **every** row in the table.

The number of rows processed can be reduced substantially by using the index-aware function ST_DWithin:

```
SELECT geom
FROM geom_table
WHERE ST_DWithin( geom, 'SRID=312;POINT(100000 200000)', 100 )
```

This query selects the same geometries, but it does it in a more efficient way. This is enabled by `ST_DWithin()` using the `&&` operator internally on an expanded bounding box of the query geometry. If there is a spatial index on `geom`, the query planner will recognize that it can use the index to reduce the number of rows scanned before calculating the distance. The spatial index allows retrieving only records with geometries whose bounding boxes overlap the expanded extent and hence which *might* be within the required distance. The actual distance is then computed to confirm whether to include the record in the result set.

For more information and examples see the PostGIS Workshop.

5.3 Examples of Spatial SQL

The examples in this section make use of a table of linear roads, and a table of polygonal municipality boundaries. The definition of the `bc_roads` table is:

```
Column     | Type              | Description
-----------+-------------------+-------------------
gid        | integer           | Unique ID
name       | character varying | Road Name
geom       | geometry          | Location Geometry (Linestring)
```

The definition of the `bc_municipality` table is:

```
Column   | Type              | Description
---------+-------------------+-------------------
gid      | integer           | Unique ID
code     | integer           | Unique ID
name     | character varying | City / Town Name
geom     | geometry          | Location Geometry (Polygon)
```

1. *What is the total length of all roads, expressed in kilometers?*

   You can answer this question with a very simple piece of SQL:

   ```
   SELECT sum(ST_Length(geom))/1000 AS km_roads FROM bc_roads;

   km_roads
   ------------------
   70842.1243039643
   ```

2. *How large is the city of Prince George, in hectares?*

   This query combines an attribute condition (on the municipality name) with a spatial calculation (of the polygon area):

   ```
   SELECT
     ST_Area(geom)/10000 AS hectares
   FROM bc_municipality
   WHERE name = 'PRINCE GEORGE';

   hectares
   ------------------
   32657.9103824927
   ```

3. *What is the largest municipality in the province, by area?*

   This query uses a spatial measurement as an ordering value. There are several ways of approaching this problem, but the most efficient is below:

   ```
   SELECT
     name,
     ST_Area(geom)/10000 AS hectares
   FROM bc_municipality
   ORDER BY hectares DESC
   LIMIT 1;

   name           | hectares
   ---------------+-----------------
   TUMBLER RIDGE  | 155020.02556131
   ```

   Note that in order to answer this query we have to calculate the area of every polygon. If we were doing this a lot it would make sense to add an area column to the table that could be indexed for performance. By ordering the results in a descending direction, and them using the PostgreSQL "LIMIT" command we can easily select just the largest value without using an aggregate function like MAX().

4. *What is the length of roads fully contained within each municipality?*

   This is an example of a "spatial join", which brings together data from two tables (with a join) using a spatial interaction ("contained") as the join condition (rather than the usual relational approach of joining on a common key):

   ```
   SELECT
     m.name,
     sum(ST_Length(r.geom))/1000 as roads_km
   FROM bc_roads AS r
   JOIN bc_municipality AS m
   ```

```
  ON ST_Contains(m.geom, r.geom)
GROUP BY m.name
ORDER BY roads_km;

name                         | roads_km
-----------------------------+------------------
SURREY                       | 1539.47553551242
VANCOUVER                    | 1450.33093486576
LANGLEY DISTRICT             | 833.793392535662
BURNABY                      | 773.769091404338
PRINCE GEORGE                | 694.37554369147
...
```

This query takes a while, because every road in the table is summarized into the final result (about 250K roads for the example table). For smaller datsets (several thousand records on several hundred) the response can be very fast.

5. *Create a new table with all the roads within the city of Prince George.*

   This is an example of an "overlay", which takes in two tables and outputs a new table that consists of spatially clipped or cut resultants. Unlike the "spatial join" demonstrated above, this query creates new geometries. An overlay is like a turbo-charged spatial join, and is useful for more exact analysis work:

```
CREATE TABLE pg_roads as
SELECT
  ST_Intersection(r.geom, m.geom) AS intersection_geom,
  ST_Length(r.geom) AS rd_orig_length,
  r.*
FROM bc_roads AS r
JOIN bc_municipality AS m
  ON ST_Intersects(r.geom, m.geom)
WHERE
  m.name = 'PRINCE GEORGE';
```

6. *What is the length in kilometers of "Douglas St" in Victoria?*

```
SELECT
  sum(ST_Length(r.geom))/1000 AS kilometers
FROM bc_roads r
JOIN bc_municipality m
  ON ST_Intersects(m.geom, r.geom
WHERE
  r.name = 'Douglas St'
  AND m.name = 'VICTORIA';

kilometers
------------------
4.89151904172838
```

7. *What is the largest municipality polygon that has a hole?*

```
SELECT gid, name, ST_Area(geom) AS area
FROM bc_municipality
WHERE ST_NRings(geom) > 1
ORDER BY area DESC LIMIT 1;

gid  | name         | area
-----+--------------+------------------
12   | SPALLUMCHEEN | 257374619.430216
```

Chapter 6

Dicas de desempenho

6.1 Pequenas tabelas de grandes geometrias

6.1.1 Descrição do problema

Versões atuais do PostgreSQL (incluindo a 8.0) sofrem de um problem no otimizador de queries quando falamos de tabelas TOAST. As tabelas TOAST são extensões utilizadas para armazenamento de grandes valores (no sentido de tamanho do dado) que não cabem normalmente nas páginas de dados (grandes blocos de texto, imagens ou geometrias complexas com muitos vértices, veja a documentação oficial para maiores informações).

Este problema ocorre se você possui tabelas com geometrias grandes, mas não muitas linhas (uma tabela dos limites todos os países europeus em alta resolução). A tabela em si, é pequena, mas utiliza muito espaço TOAST. Em nosso exemplo, a tabela em si possuía apenas 80 linhas e utilizava apenas 3 páginas de dados, mas a tabela TOAST utilizava 8225 páginas de dados.

Emita uma pesquisa onde você utiliza o operador && para pesquisa por um retângulo envolvente que bate com poucas dessas linhas. O otimizador de pesquisas ve esta tabela contendo apenas 3 páginas e 80 linhas. Como a tabela é pequena, ele estima que um scan sequencial em uma tabela tão pequena será mais rápida do que utilizar um índice, ignorando o mesmo. Geralmente esta estimativa é correta, mas em nosso caso o operador && tem que buscar todas as geometrias em disco para comparação dos retângulos envolventes, lendo todas as páginas TOAST também.

Para visualizar se você sofre com este bug, utilize um "EXPLAIN ANALYZE" na pesquisa em questão. Para maiores informações e detalhes técnicos, você pode recorrer a lista do postgres sobre desempenho: http://archives.postgresql.org/pgsql-performance/2005-02/msg00030.php

and newer thread on PostGIS https://lists.osgeo.org/pipermail/postgis-devel/2017-June/026209.html

6.1.2 Soluções

O pessoal responsável pelo PostgreSQL está tentando resolver esta questão por transformar o otimizador de pesquisas ciente das tabelas TOAST. Por enquanto, existem duas soluções:

A primeira solução é forçar o estimador de pesquisar a utilizar o índice. Emita um comando "SET enable_seqscan TO off" ao servidor antes de emitir a pesquisa. Isto força o estimador a evitar scans sequenciais sempre que possível, utilizando o índice GIST como de costume. Mas esta flag deve ser setada para cada conexão e causa o estimador a decidir mal em outros casos, portanto, você deve habilitar "SET enable_seqscan TO on;" após a pesquisa.

A segunda solução é fazer a pesquisa sequencial tão rápida quanto o estimador imagina. Isto pode ser feito criando uma coluna adicional que cacheia o retângulo envolvente e realizando as pesquisas em cima desta coluna. Em nosso exemplo, os comandos são:

```
SELECT AddGeometryColumn('myschema','mytable','bbox','4326','GEOMETRY','2');
UPDATE mytable SET bbox = ST_Envelope(ST_Force2D(geom));
```

Altere sua query para usar o operador && contra o retângulo envolvente ao invés da colunas geométrica, assim:

```
SELECT geom_column
FROM mytable
WHERE bbox && ST_SetSRID('BOX3D(0 0,1 1)'::box3d,4326);
```

Claro, se você alterar ou adicionar colunas a mytable, você deve manter o retângulo envolvente em sincronia. A forma mais transparente de fazer isto seria através de triggers, mas você também querer modificar sua aplicação para manter a coluna do retângulo envolvente atualizada or executar a query de UPDATE após cada modificação.

6.2 CLUSTERizando índices geométricos

Para tabelas que são basicamente somente-leitura, e onde um único índice é utilizado pela maioria das queries, PostgreSQL oferece o comando CLUSTER. Este comando fisicamente reordena todas as linhas da tabela assim como as do índice, assim possibilitando duas melhorias de desempenho: primeiro, para pesquisas de intervalo de índice, o número de pesquisas na tabela de dados é dramaticamente reduzido. Segundo, se seu conjunto de trabalho concentra-se em pequenos intervalos nos índices, você tem um cache mais eficiente, pois todas as informações estão divididas em poucas páginas de dados. (Sinta se convidado para ler a documentação do comando CLUSTER do manual do PostgreSQL.)

Contudo, atualmente o Postgresql não permite a clusterização de índices geométricos GIST, pois estes índices simplesmente ignoram valores nulos, retornando um erro como:

```
lwgeom=# CLUSTER my_geom_index ON my_table;
ERROR: cannot cluster when index access method does not handle null values
HINT: You may be able to work around this by marking column "geom" NOT NULL.
```

Como a HINT da mensagem te diz, você pode adicionar uma constraint "not null" na tabela para contornar o problema.

```
lwgeom=# ALTER TABLE my_table ALTER COLUMN geom SET not null;
ALTER TABLE
```

Claro, isto não vai funcionar se você de fato precisa de valores NULL em sua coluna geométrica. Adicionalmente, você deve usar o método acima para adicionar a constraint. Utilizar uma constraint do tipo CHECK como "ALTER TABLE blubb ADD CHECK (geometry is not null);" não irá funcionar.

6.3 Evitando conversão de dimensões

Algumas vezes, você tem dados que são 3D ou 4D em sua tabela, mas sempre acessa-os usando métodos OpenGIS, como ST_AsText() ou ST_AsBinary(), que somente funcionam em geometrias 2D. Eles fazem isso internamente chamando a função ST_Force2D(), que introduza um gasto extra para grandes geometrias. Para evitar este gasto extra, pode ser viável dropar essas dimensões adicionais para sempre:

```
UPDATE mytable SET geom = ST_Force2D(geom);
VACUUM FULL ANALYZE mytable;
```

Note que se você adicionou sua coluna geométrica utilizando o método AddGeometryColumn(), existirá uma constraint na dimensão da geometria. Para contornar isto, você precisará dropar a constraint também. Lembre-se de atualizar a entrada na tabela geometry_columns e recriar a constraint posteriormente.

No caso de grandes tabelas, pode ser sábio dividir este UPDATE em porções menores, restringindo o UPDATE a pequenas partes da tabela com o uso de uma cláusula WHERE sobre sua PRIMARY KEY ou outro critério, rodando um VACUUM, entre os UPDATEs. Isto reduz drasticamente a necessidade de espaço em disco temporário. Adicionalmente, se você tem geometrias de dimensões mistas, restrigir o UPDATE por "WHERE dimension(the_geom)>2" pula as geometrias que já estão em 2D.

Chapter 7

Referência do PostGIS

As funções descritas abaixo são as que um usuário do PostGIS devem precisar. Existem outras funções que são necessárias para suportar os objetos PostGIS mas que não são de uso comum pelo usuário.

Note
O PostGIS iniciou uma transição da convenção de nomenclatura existente para uma convenção em torno do SQL-MM. Como resultado, a maioria das funções que você conhece e ama foram renomeadas usando o padrão de tipo espacial (com o prefixo ST). As funções anteriores ainda existem, porém não são listadas nesta documentação onde as funções atualizadas são equivalentes. As funções que não possuem prefixo ST_ não listadas nesta documentação estão obsoletas e serão removidas em futuros lançamentos, então PAREM DE UTILIZÁ-LAS.

7.1 PostgreSQL PostGIS Geometry/Geography/Box Types

7.1.1 box2d

box2d — The type representing a 2-dimensional bounding box.

Descrição

a caixa3d é um tipo de dados postgis usados para representar a caixa enclosing de um ageometria ou conjunto de geometrias. A ST_3DExtent retorna um objeto caixa3d.

The representation contains the values `xmin, ymin, xmax, ymax`. These are the minimum and maximum values of the X and Y extents.

`box2d` objects have a text representation which looks like `BOX(1 2,5 6)`.

Comportamento Casting

Essa seção lista os casts automáticos bem como os explícitos permitidos para esse tipo de dados

Cast To	Comportamento
box3d	automático
geometry	automático

Veja também

Section 12.7

7.1.2 box3d

box3d — The type representing a 3-dimensional bounding box.

Descrição

a caixa3d é um tipo de dados postgis usados para representar a caixa enclosing de um ageometria ou conjunto de geometrias. A ST_3DExtent retorna um objeto caixa3d.

The representation contains the values `xmin, ymin, zmin, xmax, ymax, zmax`. These are the minimum and maxium values of the X, Y and Z extents.

`box3d` objects have a text representation which looks like `BOX3D(1 2 3,5 6 5)`.

Comportamento Casting

Essa seção lista os casts automáticos bem como os explícitos permitidos para esse tipo de dados

Cast To	Comportamento
box	automático
box2d	automático
geometry	automático

Veja também

Section 12.7

7.1.3 geometry

geometry — geografia é um tipo de dado espacial usado para representar uma característica no sistema de coordenada da terra-redonda.

Descrição

geografia é um tipo de dado espacial usado para representar uma característica no sistema de coordenada da terra-redonda.

All spatial operations on geometry use the units of the Spatial Reference System the geometry is in.

Comportamento Casting

Essa seção lista os casts automáticos bem como os explícitos permitidos para esse tipo de dados

Cast To	Comportamento
box	automático
box2d	automático
box3d	automático
bytea	automático
geography	automático
texto	automático

Veja também

Section 4.1, Section 4.3

7.1.4 geometry_dump

geometry_dump — A composite type used to describe the parts of complex geometry.

Descrição

`geometry_dump` is a composite data type containing the fields:

- `geom` - a geometry representing a component of the dumped geometry. The geometry type depends on the originating function.
- `path[]` - an integer array that defines the navigation path within the dumped geometry to the `geom` component. The path array is 1-based (i.e. `path[1]` is the first element.)

It is used by the `ST_Dump*` family of functions as an output type to explode a complex geometry into its constituent parts.

Veja também

Section 12.6

7.1.5 geografia

geografia — The type representing spatial features with geodetic (ellipsoidal) coordinate systems.

Descrição

geografia é um tipo de dado espacial usado para representar uma característica no sistema de coordenada da terra-redonda.

Spatial operations on the geography type provide more accurate results by taking the ellipsoidal model into account.

Comportamento Casting

Essa seção lista os casts automáticos bem como os explícitos permitidos para esse tipo de dados

Cast To	Comportamento
geometry	explícito

Veja também

Section 4.3, Section 4.3

7.2 Funções de Gestão

7.2.1 AddGeometryColumn

AddGeometryColumn — Remove uma coluna geometria de uma spatial table.

Synopsis

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, integer 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);

Descrição

Adiciona uma coluna geometria à uma table de atributos. O `schema_name` é o nome da table esquema. O `srid` deve ser um valor de referência inteiro para uma entrada na table SPATIAL_REF_SYS. O `tipo` deve ser uma string correspondente ao tipo da geometria, por exemplo: 'POLÍGONO' ou 'MULTILINSTRING'. Um erro é descartado se o esquema não existe (ou não é visível no search_path atual) ou a SRID especificada, tipo de geometria ou dimensão é inválida.

Note
Alterado: 2.0.0 Essa função não atualiza mais a geometry_columns desde que ela é a view que lê dos catálogos de sistema. Por padrão, isso não cria restrições, mas usa a construção no comportamento do tipo modificador do PostgreSQL. Então, por exemplo, construir uma coluna wgs84 POINT com essa função é equivalente a: `ALTER TABLE some_table ADD COLUMN geom geometry(Point,4326);`
Alterado: 2.0.0 Se você exige o comportamento antigo de restrições use o padrão `use_typmod`, mas configure isso para falso.

Note
Alterações: 2.0.0 Views não podem ser registradas manualmente mais em geometry_columns, porém as views construídas contra as geometrias typmod tables e usadas sem as funções wrapper irão se registrar corretamente, porque elas herdam um comportamento typmod da table column mãe. As views que usam funções geométricas que fazem outras geometrias saírem, precisarão de ser lançadas para as geometrias typmod, para essas colunas serem registradas corretamente em geometry_columns. Use Section 4.6.3.


This method implements the OGC Simple Features Implementation Specification for SQL 1.1.


This function supports 3d and will not drop the z-index.


This method supports Circular Strings and Curves.

Melhorias: 2.0.0 argumento use_typmod introduzido. Padrões para criar colunas de geometria typmod ao invés das baseadas em obstáculos.

Exemplos

```
-- Create schema to hold data
CREATE SCHEMA my_schema;
-- Create a new simple PostgreSQL table
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:
    "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     | Point        | 4326 |     2
 geom_c   | Point        | 4326 |     2
 geomcp_c | CurvePolygon | 4326 |     2
```

Veja também

DropGeometryColumn, DropGeometryTable, Section 4.6.2, Section 4.6.3

7.2.2 DropGeometryColumn

DropGeometryColumn — Remove uma coluna geometria de uma spatial table.

Synopsis

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);

Descrição

Remove uma coluna geometria de uma table espacial. Note que o schema_name precisará combinar com o campo f_table_schema da fila da table na table geometry_columns.

This method implements the OGC Simple Features Implementation Specification for SQL 1.1.

This function supports 3d and will not drop the z-index.

This method supports Circular Strings and Curves.

Note
Alterações: 2.0.0 Essa função é fornecida para compatibilidade atrasada. Desde que geometry_columns é uma view contra os sistemas catalogados, você pode derrubar uma coluna geométrica como qualquer outra table column usando `ALTERAR TABLE`

Exemplos

```
SELECT DropGeometryColumn ('my_schema','my_spatial_table','geom');
                        ----RESULT output ---
                                        dropgeometrycolumn
----------------------------------------------------
 my_schema.my_spatial_table.geom effectively removed.

-- In PostGIS 2.0+ the above is also equivalent to the standard
-- the standard alter table.  Both will deregister from geometry_columns
ALTER TABLE my_schema.my_spatial_table DROP column geom;
```

Veja também

AddGeometryColumn, DropGeometryTable, Section 4.6.2

7.2.3 DropGeometryTable

DropGeometryTable — Derruba uma table e todas suas referências em geometry_columns.

Synopsis

boolean **DropGeometryTable**(varchar table_name);
boolean **DropGeometryTable**(varchar schema_name, varchar table_name);
boolean **DropGeometryTable**(varchar catalog_name, varchar schema_name, varchar table_name);

Descrição

Derruba uma table e todas as suas referências em geometry_columns. Nota: use current_schema() nas instalações schema-aware pgsql se o esquema não for fornecido.

Note
Alterações: 2.0.0 Essa função é fornecida para compatibilidade atrasada. Desde que geometry_columns é uma view contra os sistemas catalogados, você pode derrubar uma table com colunas geométricas como qualquer outra table usando `DERRUBAR TABLE`

Exemplos

```
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;
```

Veja também

AddGeometryColumn, DropGeometryColumn, Section 4.6.2

7.2.4 Find_SRID

Find_SRID — Returns the SRID defined for a geometry column.

Synopsis

integer **Find_SRID**(varchar a_schema_name, varchar a_table_name, varchar a_geomfield_name);

Descrição

Returns the integer SRID of the specified geometry column by searching through the GEOMETRY_COLUMNS table. If the geometry column has not been properly added (e.g. with the AddGeometryColumn function), this function will not work.

Exemplos

```
SELECT Find_SRID('public', 'tiger_us_state_2007', 'geom_4269');
find_srid
----------
4269
```

Veja também

ST_SRID

7.2.5 Populate_Geometry_Columns

Populate_Geometry_Columns — Ensures geometry columns are defined with type modifiers or have appropriate spatial constraints.

Synopsis

text **Populate_Geometry_Columns**(boolean use_typmod=true);
int **Populate_Geometry_Columns**(oid relation_oid, boolean use_typmod=true);

Descrição

Assegura que as colunas geométricas são definidas com modificadores de tipo ou têm obstáculos espaciais apropriados. Isso garante que serão registrados corretamente na view `geometry_columns`. Por padrão, irá converter todas as colunas geométricas com nenhum modificador de tipo para os que têm o modificador. para obter esse comportamento antigo use `use_typmod=false`

Para compatibilidades atrasadas e necessidades espaciais como a herança das tables, onde cada table child talvez tenha um tipo geométrico diferente, a última verificação do comportamento ainda é suportada. Se você precisar do último comportamento, você tem de passar o novo argumento opcional como falso `use_typmod=false`. Quando isso for feito, as colunas geométricas serão criadas sem modificadores de tipo, mas terão 3 obstáculos definidos. Isso significa que cada coluna geométrica pertencente a uma table tem, pelo menos, três obstáculos:

- `enforce_dims_the_geom` - assegura que toda geometria tenha a mesma dimensão (veja ST_NDims)
- `enforce_geotype_the_geom` - assegura que toda geometria seja do mesmo tipo (veja Tipo de geometria)
- `enforce_srid_the_geom` - assegura que toda geometria tenha a mesma projeção (veja ST_SRID)

Se uma table `oid` é fornecida, essa função tenta determinar a srid, a dimensão e o tipo geométrico de todas as colunas geométricas na table, adicionando restrições se necessário. Se for bem-sucedido, uma fila apropriada é inserida na table geometry_columns, senão, a exceção é pega e uma notificação de erro surge, descrevendo o problema.

Se o `oid` de uma view é fornecido, como com uma table oid, essa função tenta determinar a srid, dimensão e tipo de todas as geometrias na view, inserindo entradas apropriadas na table `geometry_columns`, mas nada é feito para executar obstáculos.

A variante sem parâmetro é um simples wrapper para a variante parametrizada que trunca primeiro e repopula a table geometry_columns para cada table espacial e view no banco de dados, adicionando obstáculos espaciais para tables onde são apropriados. Isso retorna um resumo do número de colunas geométricas detectadas no banco de dados e o número que foi inserido na table `geometry_columns`. A versão parametrizada retorna, simplesmente, o número de filas inseridas na table `geometry_columns`.

Disponibilidade: 1.4.0

Alterações: 2.0.0 Por padrão, utilize modificadores de tipo ao invés de verificar restrições para restringir os tipos de geometria. Você pode verificar restrições de comportamento ao invés de usar o novo `use_typmod` e configurá-lo para falso.

Melhorias: 2.0.0 `use_typmod` argumento opcional foi introduzido, permitindo controlar se as colunas forem criadas com modificadores de tipo ou com verificação de restrições.

Exemplos

```
CREATE TABLE public.myspatial_table(gid serial, geom geometry);
INSERT INTO myspatial_table(geom) VALUES(ST_GeomFromText('LINESTRING(1 2, 3 4)',4326) );
-- This will now use typ modifiers.  For this to work, there must exist data
SELECT Populate_Geometry_Columns('public.myspatial_table'::regclass);

populate_geometry_columns
--------------------------
                         1


\d myspatial_table

                                  Table "public.myspatial_table"
 Column |           Type            |                          Modifiers
```

```
--------+--------------------------+-------------------------------------------------------------------

 gid    | integer                  | not null default nextval('myspatial_table_gid_seq':: ↩
    regclass)
 geom   | geometry(LineString,4326) |
```

```
-- This will change the geometry columns to use constraints if they are not typmod or have  ↩
   constraints already.
--For this to work, there must exist data
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)
```

7.2.6 UpdateGeometrySRID

UpdateGeometrySRID — Updates the SRID of all features in a geometry column, and the table metadata.

Synopsis

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);

Descrição

Atualiza a SRID de todas as características em uma coluna geométrica, atualizando restrições e referências na geometry_columns. Nota: use current_schema() nas instalações schema-aware pgsql se o esquema não for fornecido.

 This function supports 3d and will not drop the z-index.

 This method supports Circular Strings and Curves.

Exemplos

Insert geometries into roads table with a SRID set already using EWKT format:

```
COPY roads (geom) FROM STDIN;
SRID=4326;LINESTRING(0 0, 10 10)
SRID=4326;LINESTRING(10 10, 15 0)
\.
```

Isso irá alterar a srid das roads tables para 4326 de qualquer coisa que tenha sido antes

```
SELECT UpdateGeometrySRID('roads','geom',4326);
```

O exemplo anterior é equivalente a esta declaração DDL

```
ALTER TABLE roads
  ALTER COLUMN geom TYPE geometry(MULTILINESTRING, 4326)
    USING ST_SetSRID(geom,4326);
```

Se você obteve a projeção errada (ou comprou como desconhecido) no carregamento e quer transformar para mercartor, tudo de uma vez, você pode fazer isso com DDL, mas não existe uma função de gestão equivalente do PostGIS.

```
ALTER TABLE roads
 ALTER COLUMN geom TYPE geometry(MULTILINESTRING, 3857) USING ST_Transform(ST_SetSRID(geom ↩
     ,4326),3857) ;
```

Veja também

UpdateRasterSRID, ST_SetSRID, ST_Transform

7.3 Construtores de geometria

7.3.1 ST_GeomCollFromText

ST_GeomCollFromText — Creates a GeometryCollection or Multi* geometry from a set of geometries.

Synopsis

geometry **ST_Collect**(geometry g1, geometry g2);
geometry **ST_Collect**(geometry[] g1_array);
geometry **ST_Collect**(geometry set g1field);

Descrição

Collects geometries into a geometry collection. The result is either a Multi* or a GeometryCollection, depending on whether the input geometries have the same or different types (homogeneous or heterogeneous). The input geometries are left unchanged within the collection.

Variant 1: accepts two input geometries

Variant 2: accepts an array of geometries

Variant 3: aggregate function accepting a rowset of geometries.

Note
If any of the input geometries are collections (Multi* or GeometryCollection) ST_Collect returns a GeometryCollection (since that is the only type which can contain nested collections). To prevent this, use ST_Dump in a subquery to expand the input collections to their atomic elements (see example below).

Note
ST_Collect and ST_Union appear similar, but in fact operate quite differently. ST_Collect aggregates geometries into a collection without changing them in any way. ST_Union geometrically merges geometries where they overlap, and splits linestrings at intersections. It may return single geometries when it dissolves boundaries.

Disponibilidade: 1.4.0 - ST_MakeLine(geomarray) foi introduzida. A ST_MakeLine agrega funções que foram melhoradas para lidar com mais pontos mais rápido.

 This function supports 3d and will not drop the z-index.

 This method supports Circular Strings and Curves.

Exemplos - Uso XLink

Collect 2D points.

```
SELECT ST_AsText( ST_Collect( ST_GeomFromText('POINT(1 2)'),
        ST_GeomFromText('POINT(-2 3)') ));

st_astext
----------
MULTIPOINT((1 2),(-2 3))
```

Collect 3D points.

```
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)
```

Collect curves.

```
SELECT ST_AsText( ST_Collect( 'CIRCULARSTRING(220268 150415,220227 150505,220227 150406)',
                'CIRCULARSTRING(220227 150406,2220227 150407,220227 150406)'));

                st_astext
-----------------------------------------------------------------------------------------
MULTICURVE(CIRCULARSTRING(220268 150415,220227 150505,220227 150406),
 CIRCULARSTRING(220227 150406,2220227 150407,220227 150406))
```

Exemplos: Utilizando versão banco de dados

Using an array constructor for a subquery.

```
SELECT ST_Collect( ARRAY( SELECT geom FROM sometable ) );
```

Using an array constructor for values.

```
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))
```

Exemplos: Versão espacial agregada

Creating multiple collections by grouping geometries in a table.

```
SELECT stusps, ST_Collect(f.geom) as geom
         FROM (SELECT stusps, (ST_Dump(geom)).geom As geom
                              FROM
                              somestatetable ) As f
        GROUP BY stusps
```

Veja também

ST_Dump, ST_Union

7.3.2 ST_LineFromMultiPoint

ST_LineFromMultiPoint — Cria uma linestring de um multiponto geométrico.

Synopsis

geometria **ST_LineFromMultiPoint**(geometria ummultiponto);

Descrição

Cria uma LineString de uma geometria MultiPointo.

Use ST_MakeLine to create lines from Point or LineString inputs.

 This function supports 3d and will not drop the z-index.

Examples

Cria uma LineString de uma geometria MultiPointo.

```
SELECT ST_AsEWKT(  ST_LineFromMultiPoint('MULTIPOINT(1 2 3, 4 5 6, 7 8 9)')  ));

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

Veja também

ST_AsEWKT, ST_AsKML

7.3.3 ST_MakeEnvelope

ST_MakeEnvelope — Cria um polígono retangular formado a partir dos mínimos e máximos dados. Os valores de entrada devem ser em SRS especificados pelo SRID.

Synopsis

geometry **ST_MakeEnvelope**(double precision xmin, double precision ymin, double precision xmax, double precision ymax, integer srid=unknown);

Descrição

Cria um polígono retangular formado a partir do mínimo e máximo, pela dada shell. Os valores de entradas devem ser SRS especificados pelo SRID. Se nenhum SRID for especificado o sistema de referência espacial desconhecido é assumido

Disponibilidade: 1.5

Melhorias: 2.0: Habilidade para especificar um pacote sem especificar um SRID foi introduzida.

Exemplo: Construindo um polígono bounding box

```
SELECT ST_AsText( ST_MakeEnvelope(10, 10, 11, 11, 4326) );

st_asewkt
-----------
POLYGON((10 10, 10 11, 11 11, 11 10, 10 10))
```

Veja também

ST_MakePoint, ST_MakePoint, ST_Point, ST_SRID

7.3.4 ST_MakeLine

ST_MakeLine — Cria uma Linestring de ponto, multiponto ou linha das geometrias.

Synopsis

geometry **ST_MakeLine**(geometry geom1, geometry geom2);
geometry **ST_MakeLine**(geometry[] geoms_array);
geometry **ST_MakeLine**(geometry set geoms);

Descrição

Creates a LineString containing the points of Point, MultiPoint, or LineString geometries. Other geometry types cause an error.

Variant 1: accepts two input geometries

Variant 2: accepts an array of geometries

Variant 3: aggregate function accepting a rowset of geometries. To ensure the order of the input geometries use `ORDER BY` in the function call, or a subquery with an `ORDER BY` clause.

Repeated nodes at the beginning of input LineStrings are collapsed to a single point. Repeated points in Point and MultiPoint inputs are not collapsed. ST_RemoveRepeatedPoints can be used to collapse repeated points from the output LineString.

 This function supports 3d and will not drop the z-index.

Disponibilidad: 2.0.0 - Suporte para elementos de entrada linestring foi introduzido

Disponibilidad: 2.0.0 - Suporte para elementos de entrada linestring foi introduzido

Disponibilidade: 1.4.0 - ST_MakeLine(geomarray) foi introduzida. A ST_MakeLine agrega funções que foram melhoradas para lidar com mais pontos mais rápido.

Exemplos: Utilizando versão banco de dados

Create a line composed of two points.

```
SELECT ST_AsText( ST_MakeLine(ST_Point(1,2), ST_Point(3,4)) );

          st_astext
---------------------
 LINESTRING(1 2,3 4)
```

Cria uma CAIXA2D definida pelos pontos 2 3D dados das geometrias.

```
SELECT ST_AsEWKT( ST_MakeLine(ST_MakePoint(1,2,3), ST_MakePoint(3,4,5) ));

                st_asewkt
--------------------------
 LINESTRING(1 2 3,3 4 5)
```

Cria uma Linestring de ponto, multiponto ou linha das geometrias.

```
select ST_AsText( ST_MakeLine( 'LINESTRING(0 0, 1 1)', 'LINESTRING(2 2, 3 3)' ) );

          st_astext
-----------------------------
 LINESTRING(0 0,1 1,2 2,3 3)
```

Exemplos: Utilizando versão banco de dados

Create a line from an array formed by a subquery with ordering.

```
SELECT ST_MakeLine( ARRAY( SELECT ST_Centroid(geom) FROM visit_locations ORDER BY  ↩
    visit_time) );
```

Create a 3D line from an array of 3D 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)
```

Exemplos: Versão espacial agregada

Esse exemplo pega uma sequência de pontos do GPS e cria um relato para cada torre gps onde o campo geométrico é uma line string composta com os pontos do gps na ordem da viagem.

Using aggregate `ORDER BY` provides a correctly-ordered LineString.

```
SELECT gps.track_id, ST_MakeLine(gps.geom ORDER BY gps_time) As geom
        FROM gps_points As gps
        GROUP BY track_id;
```

Prior to PostgreSQL 9, ordering in a subquery can be used. However, sometimes the query plan may not respect the order of the subquery.

```
SELECT gps.track_id, ST_MakeLine(gps.geom) As geom
        FROM ( SELECT track_id, gps_time, geom
                        FROM gps_points ORDER BY track_id, gps_time ) As gps
        GROUP BY track_id;
```

Veja também

ST_RemoveRepeatedPoints, ST_AsText, ST_GeomFromText, ST_MakePoint

7.3.5 ST_MakePoint

ST_MakePoint — Creates a 2D, 3DZ or 4D Point.

Synopsis

geometria **ST_Point**(float x_lon, float y_lat);

geometry **ST_MakePointM**(float x, float y, float m);

geometry **ST_MakePoint**(double precision x, double precision y, double precision z, double precision m);

Descrição

Cria uma CAIXA2D definida pelos pontos dados das geometrias.

Use ST_MakePointM to make points with XYM coordinates.

While not OGC-compliant, `ST_MakePoint` is faster and more precise than ST_GeomFromText and ST_PointFromText. It is also easier to use for numeric coordinate values.

Note
For geodetic coordinates, `X` is longitude and `Y` is latitude

---

 This function supports 3d and will not drop the z-index.

Examples

```
--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
```

Veja também

ST_GeomFromText, ST_PointFromText, ST_SetSRID, ST_MakePointM

7.3.6 ST_MakePointM

ST_MakePointM — Cria um ponto com uma coordenada x y e medida.

Synopsis

geometry **ST_MakePointM**(float x, float y, float m);

Descrição

Cria um ponto com uma coordenada x y e medida.

Use ST_MakePoint to make points with XY, XYZ, or XYZM coordinates.

Note
For geodetic coordinates, `X` is longitude and `Y` is latitude

Examples

Note
ST_AsEWKT is used for text output because ST_AsText does not support M values.

Create point with unknown SRID.

```
SELECT ST_AsEWKT(  ST_MakePointM(-71.1043443253471, 42.3150676015829, 10)  );

                                  st_asewkt
-----------------------------------------------
 POINTM(-71.1043443253471 42.3150676015829 10)
```

Cria um ponto com uma coordenada x y e medida.

```
SELECT ST_AsEWKT( ST_SetSRID(  ST_MakePointM(-71.104, 42.315, 10),  4326));

                                                st_asewkt
----------------------------------------------------------
SRID=4326;POINTM(-71.104 42.315 10)
```

Get measure of created point.

```
SELECT ST_M(  ST_MakePointM(-71.104, 42.315, 10)  );

result
-------
10
```

Veja também

ST_AsEWKT, ST_MakePoint, ST_SetSRID

7.3.7 ST_MakePolygon

ST_MakePolygon — Creates a Polygon from a shell and optional list of holes.

Synopsis

geometry **ST_MakePolygon**(geometry linestring);

geometry **ST_MakePolygon**(geometry outerlinestring, geometry[] interiorlinestrings);

Descrição

Cria uma polígono formado pela dada shell. As geometrias de entrada devem ser LINESTRINGS fechadas.

Variant 1: Accepts one shell LineString.

Variant 2: Accepts a shell LineString and an array of inner (hole) LineStrings. A geometry array can be constructed using the PostgreSQL array_agg(), ARRAY[] or ARRAY() constructs.

Note
Essa função não aceitará uma MULTILINESTRING. Use ST_LineMerge ou ST_Dump para gerar line strings.

 This function supports 3d and will not drop the z-index.

Exemplos: Utilizando versão banco de dados

Cria uma LineString de uma string Encoded Polyline.

```
SELECT ST_MakePolygon( ST_GeomFromText('LINESTRING(75 29,77 29,77 29, 75 29)'));
```

Create a Polygon from an open LineString, using ST_StartPoint and ST_AddPoint to close it.

```
SELECT ST_MakePolygon( ST_AddPoint(foo.open_line, ST_StartPoint(foo.open_line)) )
FROM (
  SELECT ST_GeomFromText('LINESTRING(75 29,77 29,77 29, 75 29)') As open_line) As foo;
```

Cria uma LineString de uma string Encoded Polyline.

```
SELECT ST_AsEWKT( ST_MakePolygon( '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))
```

Create a Polygon from a LineString with measures

```
SELECT ST_AsEWKT( ST_MakePolygon( '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))
```

Exemplos: estrutura de dentro e estrutura de fora

Construir um donut com um buraco de formiga

```
SELECT ST_MakePolygon( ST_ExteriorRing( ST_Buffer(ring.line,10)),
        ARRAY[  ST_Translate(ring.line, 1, 1),
                ST_ExteriorRing(ST_Buffer(ST_Point(20,20),1)) ]
        )
FROM (SELECT ST_ExteriorRing(
        ST_Buffer(ST_Point(10,10),10,10)) AS line ) AS ring;
```

Create a set of province boundaries with holes representing lakes. The input is a table of province Polygons/MultiPolygons and a table of water linestrings. Lines forming lakes are determined by using ST_IsClosed. The province linework is extracted by using ST_Boundary. As required by `ST_MakePolygon`, the boundary is forced to be a single LineString by using ST_LineMerge. (However, note that if a province has more than one region or has islands this will produce an invalid polygon.) Using a LEFT JOIN ensures all provinces are included even if they have no lakes.

Note
A construção CASE é usada porque sustentar uma coleção de nulos em ST_MakePolygon resulta em NULO.

```
SELECT p.gid, p.province_name,
        CASE WHEN array_agg(w.geom) IS NULL
        THEN p.geom
        ELSE  ST_MakePolygon( ST_LineMerge(ST_Boundary(p.geom)),
                        array_agg(w.geom)) END
FROM
        provinces p LEFT JOIN waterlines w
                ON (ST_Within(w.geom, p.geom) AND ST_IsClosed(w.geom))
GROUP BY p.gid, p.province_name, p.geom;
```

Another technique is to utilize a correlated subquery and the ARRAY() constructor that converts a row set to an array.

```
SELECT p.gid,  p.province_name,
    CASE WHEN EXISTS( SELECT w.geom
        FROM waterlines w
        WHERE ST_Within(w.geom, p.geom)
        AND ST_IsClosed(w.geom))
    THEN ST_MakePolygon(
        ST_LineMerge(ST_Boundary(p.geom)),
        ARRAY( SELECT w.geom
            FROM waterlines w
            WHERE ST_Within(w.geom, p.geom)
            AND ST_IsClosed(w.geom)))
    ELSE p.geom
    END AS geom
FROM provinces p;
```

Veja também

ST_BuildArea ST_Polygon

7.3.8 ST_Point

ST_Point — Creates a Point with X, Y and SRID values.

Synopsis

geometria **ST_Point**(float x_lon, float y_lat);

geometry **ST_MakePointM**(float x, float y, float m);

Descrição

Returns a Point with the given X and Y coordinate values. This is the SQL-MM equivalent for ST_MakePoint that takes just X and Y.

Note
For geodetic coordinates, `X` is longitude and `Y` is latitude

Enhanced: 3.2.0 srid as an extra optional argument was added. Older installs require combining with ST_SetSRID to mark the srid on the geometry.

 This method implements the SQL/MM specification. SQL-MM 3: 6.1.2

Exemplos: Geometria

```
SELECT ST_Point( -71.104, 42.315);
```

```
SELECT ST_SetSRID(ST_Point( -71.104, 42.315),4326);
```

New in 3.2.0: With SRID specified

```
SELECT ST_Point( -71.104, 42.315, 4326);
```

Exemplos: Geografia

Pre-PostGIS 3.2 syntax

```
SELECT CAST( ST_SetSRID(ST_Point( -71.104, 42.315), 4326) AS geography);
```

3.2 and on you can include the srid

```
SELECT CAST( ST_Point( -71.104, 42.315, 4326) AS geography);
```

PostgreSQL also provides the `::` short-hand for casting

```
SELECT ST_Point( -71.104, 42.315, 4326)::geography;
```

If the point coordinates are not in a geodetic coordinate system (such as WGS84), then they must be reprojected before casting to a geography. In this example a point in Pennsylvania State Plane feet (SRID 2273) is projected to WGS84 (SRID 4326).

```
SELECT ST_Transform(ST_SetSRID( ST_Point( 3637510, 3014852 ), 2273), 4326)::geography;
```

Veja também

Section 4.3, ST_MakePoint, ST_SetSRID, ST_Transform, ST_Point, ST_Point, ST_Point

7.3.9 ST_Point

ST_Point — Creates a Point with X, Y, Z and SRID values.

Synopsis

geometry **ST_MakePoint**(double precision x, double precision y, double precision z, double precision m);

Descrição

Retorna uma ST_Point com os valores de coordenada dados. Heterônimo OGC para ST_MakePoint.

Enhanced: 3.2.0 srid as an extra optional argument was added. Older installs require combining with ST_SetSRID to mark the srid on the geometry.

Examples

```
SELECT ST_PointZ(-71.104, 42.315, 3.4, 4326)
```

```
SELECT ST_PointZ(-71.104, 42.315, 3.4, srid =
> 4326)
```

```
SELECT ST_PointZ(-71.104, 42.315, 3.4)
```

Veja também

ST_MakePoint, ST_PointFromText, ST_SetSRID, ST_MakePointM

7.3.10 ST_Point

ST_Point — Creates a Point with X, Y, M and SRID values.

Synopsis

geometry **ST_PointM**(float x, float y, float m, integer srid=unknown);

Descrição

Retorna uma ST_Point com os valores de coordenada dados. Heterônimo OGC para ST_MakePoint.

Enhanced: 3.2.0 srid as an extra optional argument was added. Older installs require combining with ST_SetSRID to mark the srid on the geometry.

Examples

```
SELECT ST_PointM(-71.104, 42.315, 3.4, 4326)
```

```
SELECT ST_PointM(-71.104, 42.315, 3.4, srid =
> 4326)
```

```
SELECT ST_PointM(-71.104, 42.315, 3.4)
```

Veja também

ST_MakePoint, ST_PointFromText, ST_SetSRID, ST_MakePointM

7.3.11 ST_Point

ST_Point — Creates a Point with X, Y, Z, M and SRID values.

Synopsis

geometry **ST_MakeEnvelope**(double precision xmin, double precision ymin, double precision xmax, double precision ymax, integer srid=unknown);

Descrição

Retorna uma ST_Point com os valores de coordenada dados. Heterônimo OGC para ST_MakePoint.

Enhanced: 3.2.0 srid as an extra optional argument was added. Older installs require combining with ST_SetSRID to mark the srid on the geometry.

Examples

```
SELECT ST_PointZM(-71.104, 42.315, 3.4, 4.5, 4326)
```

```
SELECT ST_PointZM(-71.104, 42.315, 3.4, 4.5, srid =
> 4326)
```

```
SELECT ST_PointZM(-71.104, 42.315, 3.4, 4.5)
```

Veja também

ST_MakePoint, ST_Point, ST_Point, ST_Point, ST_SetSRID

7.3.12 ST_Polygon

ST_Polygon — Creates a Polygon from a LineString with a specified SRID.

Synopsis

geometry **ST_Polygon**(geometry aLineString, integer srid);

Descrição

Returns a polygon built from the given LineString and sets the spatial reference system from the `srid`.

ST_Polygon is similar to ST_MakePolygon Variant 1 with the addition of setting the SRID.

, ST_MakePoint, ST_SetSRID

Note
Essa função não aceitará uma MULTILINESTRING. Use ST_LineMerge ou ST_Dump para gerar line strings.

This method implements the OGC Simple Features Implementation Specification for SQL 1.1.

This method implements the SQL/MM specification. SQL-MM 3: 8.3.2

This function supports 3d and will not drop the z-index.

Examples

Create a 2D polygon.

```
SELECT ST_AsText( ST_Polygon('LINESTRING(75 29, 77 29, 77 29, 75 29)'::geometry, 4326) );

-- result --
POLYGON((75 29, 77 29, 77 29, 75 29))
```

Create a 3D polygon.

```
SELECT ST_AsEWKT( ST_Polygon( ST_GeomFromEWKT('LINESTRING(75 29 1, 77 29 2, 77 29 3, 75 29 ↩
    1)'), 4326) );

-- result --
SRID=4326;POLYGON((75 29 1, 77 29 2, 77 29 3, 75 29 1))
```

Veja também

ST_AsEWKT, ST_AsText, ST_GeomFromEWKT, ST_GeomFromText, ST_LineMerge, ST_MakePolygon

7.3.13 ST_MakeEnvelope

ST_MakeEnvelope — Creates a rectangular Polygon in Web Mercator (SRID:3857) using the XYZ tile system.

Synopsis

geometry **ST_MakePoint**(double precision x, double precision y, double precision z, double precision m);

Descrição

Creates a rectangular Polygon giving the extent of a tile in the XYZ tile system. The tile is specifed by the zoom level Z and the XY index of the tile in the grid at that level. Can be used to define the tile bounds required by ST_AsMVTGeom to convert geometry into the MVT tile coordinate space.

By default, the tile envelope is in the Web Mercator coordinate system (SRID:3857) using the standard range of the Web Mercator system (-20037508.342789, 20037508.342789). This is the most common coordinate system used for MVT tiles. The optional `bounds` parameter can be used to generate tiles in any coordinate system. It is a geometry that has the SRID and extent of the "Zoom Level zero" square within which the XYZ tile system is inscribed.

The optional `margin` parameter can be used to expand a tile by the given percentage. E.g. `margin=0.125` expands the tile by 12.5%, which is equivalent to buffer=512 when the tile extent size is 4096, as used in ST_AsMVTGeom. This is useful to create a tile buffer to include data lying outside of the tile's visible area, but whose existence affects the tile rendering. For example, a

city name (a point) could be near an edge of a tile, so its label should be rendered on two tiles, even though the point is located in the visible area of just one tile. Using expanded tiles in a query will include the city point in both tiles. Use a negative value to shrink the tile instead. Values less than -0.5 are prohibited because that would eliminate the tile completely. Do not specify a margin when using with `ST_AsMVTGeom`. See the example for ST_AsMVT.

Melhorias: 2.0.0 parâmetro opcional padrão srid adicionado.

Disponibilidade: 2.1.0

Exemplo: Construindo um polígono bounding box

```
SELECT ST_AsText( ST_TileEnvelope(2, 1, 1) );

 st_astext
------------------------------
 POLYGON((-10018754.1713945 0,-10018754.1713945 10018754.1713945,0 10018754.1713945,0  ↩
     0,-10018754.1713945 0))

SELECT ST_AsText( ST_TileEnvelope(3, 1, 1, ST_MakeEnvelope(-180, -90, 180, 90, 4326) ) );

                      st_astext
------------------------------------------------------
 POLYGON((-135 45,-135 67.5,-90 67.5,-90 45,-135 45))
```

Veja também

ST_MakeEnvelope

7.3.14 ST_HexagonGrid

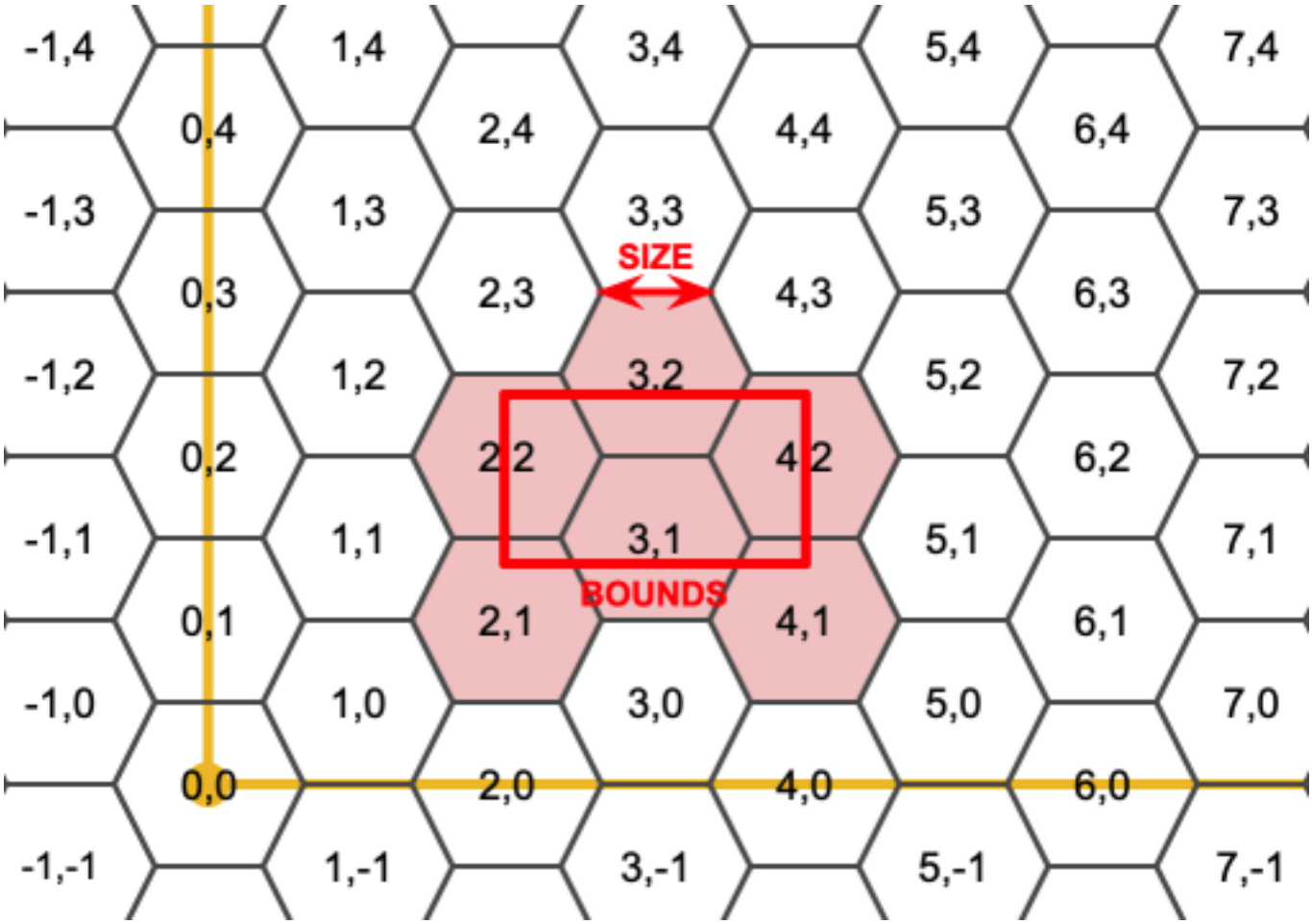
ST_HexagonGrid — Returns a set of hexagons and cell indices that completely cover the bounds of the geometry argument.

Synopsis

geometria **ST_Point**(float x_lon, float y_lat);

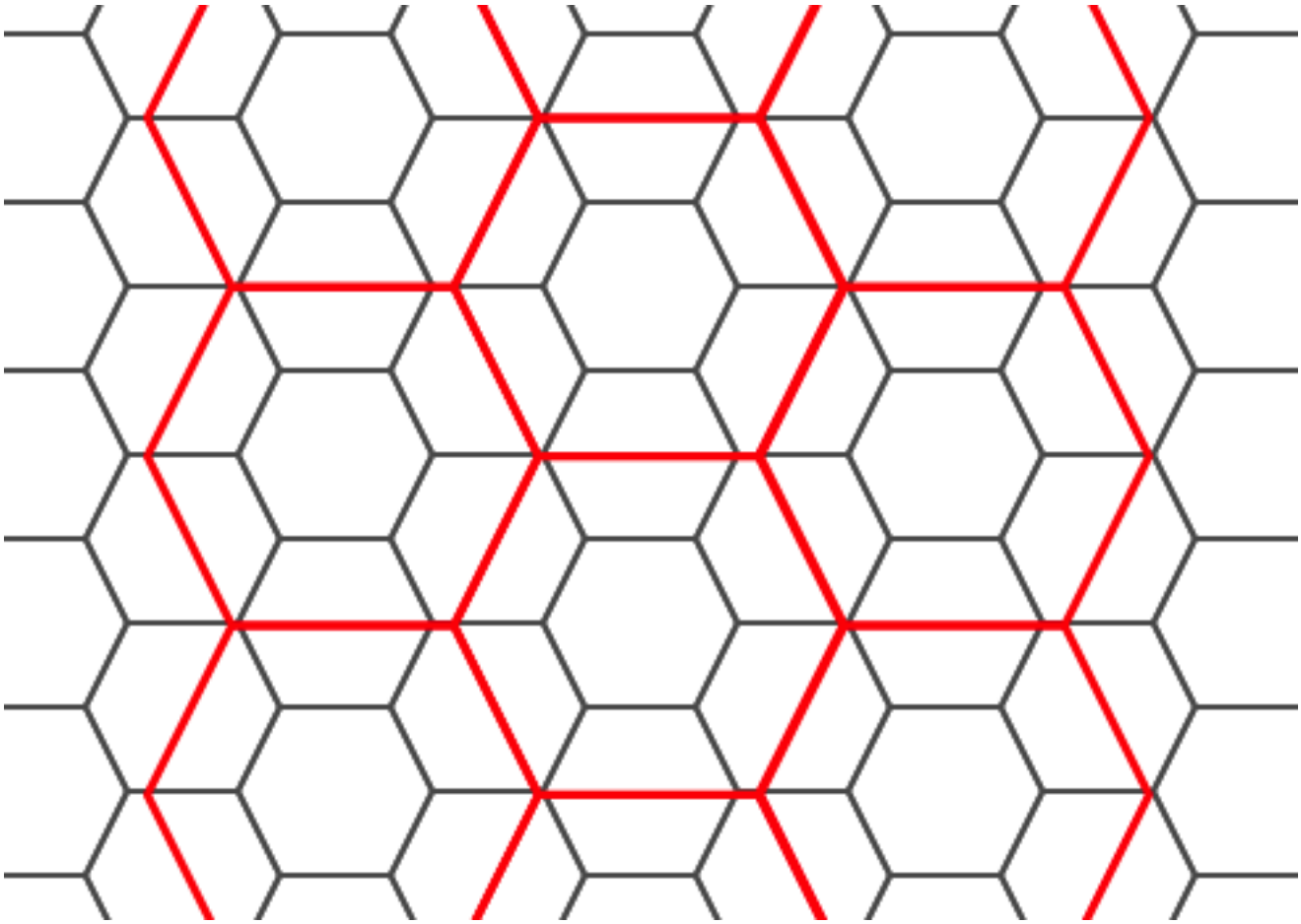
Descrição

Starts with the concept of a hexagon tiling of the plane. (Not a hexagon tiling of the globe, this is not the H3 tiling scheme.) For a given planar SRS, and a given edge size, starting at the origin of the SRS, there is one unique hexagonal tiling of the plane, Tiling(SRS, Size). This function answers the question: what hexagons in a given Tiling(SRS, Size) overlap with a given bounds.



The SRS for the output hexagons is the SRS provided by the bounds geometry.

Doubling or tripling the edge size of the hexagon generates a new parent tiling that fits with the origin tiling. Unfortunately, it is not possible to generate parent hexagon tilings that the child tiles perfectly fit inside.



Disponibilidade: 2.1.0

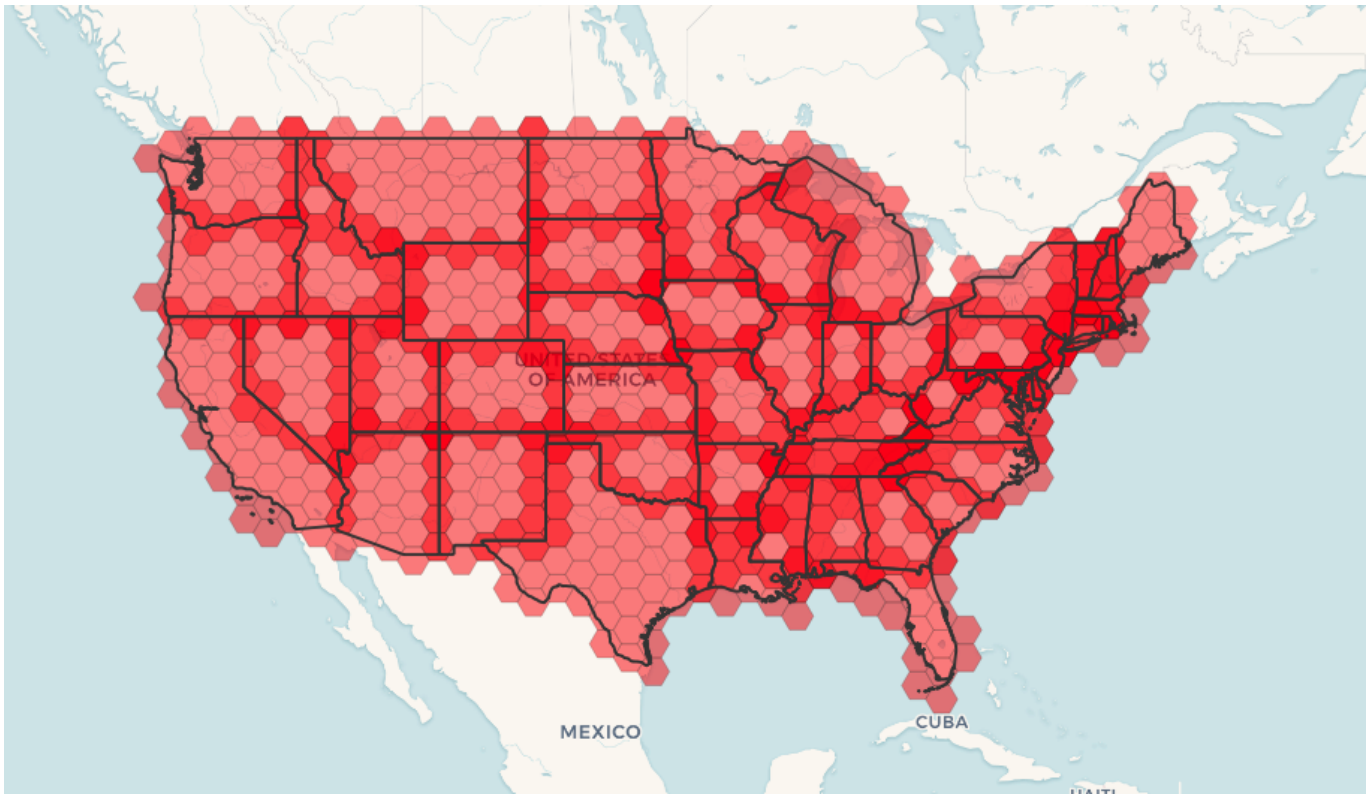
Exemplos: Utilizando versão banco de dados

To do a point summary against a hexagonal tiling, generate a hexagon grid using the extent of the points as the bounds, then spatially join to that grid.

```
SELECT COUNT(*), hexes.geom
FROM
    ST_HexagonGrid(
        10000,
        ST_SetSRID(ST_EstimatedExtent('pointtable', 'geom'), 3857)
    ) AS hexes
    INNER JOIN
    pointtable AS pts
    ON ST_Intersects(pts.geom, hexes.geom)
GROUP BY hexes.geom;
```

Exemplo: Construindo um polígono bounding box

If we generate a set of hexagons for each polygon boundary and filter out those that do not intersect their hexagons, we end up with a tiling for each polygon.



Tiling states results in a hexagon coverage of each state, and multiple hexagons overlapping at the borders between states.

Note

The LATERAL keyword is implied for set-returning functions when referring to a prior table in the FROM list. So CROSS JOIN LATERAL, CROSS JOIN, or just plain , are equivalent constructs for this example.

```
SELECT admin1.gid, hex.geom
FROM
    admin1
    CROSS JOIN
    ST_HexagonGrid(100000, admin1.geom) AS hex
WHERE
    adm0_a3 = 'USA'
    AND
    ST_Intersects(admin1.geom, hex.geom)
```

Veja também

ST_EstimatedExtent, ST_MakePoint, ST_Point, ST_SRID

7.3.15 ST_Hexagon

ST_Hexagon — Returns a single hexagon, using the provided edge size and cell coordinate within the hexagon grid space.

Synopsis

geometry **ST_MakePoint**(double precision x, double precision y, double precision z, double precision m);

Descrição

Uses the same hexagon tiling concept as ST_HexagonGrid, but generates just one hexagon at the desired cell coordinate. Optionally, can adjust origin coordinate of the tiling, the default origin is at 0,0.

Hexagons are generated with no SRID set, so use ST_SetSRID to set the SRID to the one you expect.

Disponibilidade: 2.1.0

Example: Creating a hexagon at the origin

```
SELECT ST_AsText(ST_SetSRID(ST_Hexagon(1.0, 0, 0), 3857));

POLYGON((-1 0,-0.5
         -0.866025403784439,0.5
         -0.866025403784439,1
         0,0.5
         0.866025403784439,-0.5
         0.866025403784439,-1 0))
```

Veja também

ST_MakeEnvelope, ST_MakePoint, ST_SetSRID

7.3.16 ST_SquareGrid

ST_SquareGrid — Returns a set of grid squares and cell indices that completely cover the bounds of the geometry argument.

Synopsis

geometria **ST_Point**(float x_lon, float y_lat);

Descrição

Starts with the concept of a square tiling of the plane. For a given planar SRS, and a given edge size, starting at the origin of the SRS, there is one unique square tiling of the plane, Tiling(SRS, Size). This function answers the question: what grids in a given Tiling(SRS, Size) overlap with a given bounds.

The SRS for the output squares is the SRS provided by the bounds geometry.

Doubling or edge size of the square generates a new parent tiling that perfectly fits with the original tiling. Standard web map tilings in mercator are just powers-of-two square grids in the mercator plane.

Disponibilidade: 2.1.0

Exemplo: Construindo um polígono bounding box

The grid will fill the whole bounds of the country, so if you want just squares that touch the country you will have to filter afterwards with ST_Intersects.

```
WITH grid AS (
SELECT (ST_SquareGrid(1, ST_Transform(geom,4326))).*
FROM admin0 WHERE name = 'Canada'
)
  SELEcT ST_AsText(geom)
  FROM grid
```

Example: Counting points in squares (using single chopped grid)

To do a point summary against a square tiling, generate a square grid using the extent of the points as the bounds, then spatially join to that grid. Note the estimated extent might be off from actual extent, so be cautious and at very least make sure you've analyzed your table.

```
SELECT COUNT(*), squares.geom
    FROM
    pointtable AS pts
    INNER JOIN
    ST_SquareGrid(
        1000,
        ST_SetSRID(ST_EstimatedExtent('pointtable', 'geom'), 3857)
    ) AS squares
    ON ST_Intersects(pts.geom, squares.geom)
    GROUP BY squares.geom
```

Example: Counting points in squares using set of grid per point

This yields the same result as the first example but will be slower for a large number of points

```
SELECT COUNT(*), squares.geom
    FROM
    pointtable AS pts
    INNER JOIN
    ST_SquareGrid(
        1000,
       pts.geom
    ) AS squares
    ON ST_Intersects(pts.geom, squares.geom)
    GROUP BY squares.geom
```

Veja também

ST_MakeEnvelope, ST_Point, ST_SetSRID, ST_SRID

7.3.17 ST_Square

ST_Square — Returns a single square, using the provided edge size and cell coordinate within the square grid space.

Synopsis

geometry **ST_MakePoint**(double precision x, double precision y, double precision z, double precision m);

Descrição

Uses the same square tiling concept as ST_SquareGrid, but generates just one square at the desired cell coordinate. Optionally, can adjust origin coordinate of the tiling, the default origin is at 0,0.

Squares are generated with no SRID set, so use ST_SetSRID to set the SRID to the one you expect.

Disponibilidade: 2.1.0

Example: Creating a square at the origin

```
SELECT ST_AsText(ST_SetSRID(ST_Square(1.0, 0, 0), 3857));

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

Veja também

ST_MakeEnvelope, ST_SquareGrid, ST_Hexagon

7.3.18 ST_Letters

ST_Letters — Returns the input letters rendered as geometry with a default start position at the origin and default text height of 100.

Synopsis

geometry **ST_Letters**(text letters, json font);

Descrição

Uses a built-in font to render out a string as a multipolygon geometry. The default text height is 100.0, the distance from the bottom of a descender to the top of a capital. The default start position places the start of the baseline at the origin. Over-riding the font involves passing in a json map, with a character as the key, and base64 encoded TWKB for the font shape, with the fonts having a height of 1000 units from the bottom of the descenders to the tops of the capitals.

The text is generated at the origin by default, so to reposition and resize the text, first apply the `ST_Scale` function and then apply the `ST_Translate` function.

Disponibilidade: 2.1.0

Exemplo: Construindo um polígono bounding box

```
SELECT ST_AsText(ST_Letters('Yo'), 1);
```

Letters generated by ST_Letters

Example: Scaling and moving words

```
SELECT ST_Translate(ST_Scale(ST_Letters('Yo'), 10, 10), 100,100);
```

Veja também

ST_AsTWKB, ST_Scale, ST_Translate

7.4 Acessors de Geometria

7.4.1 Tipo de geometria

Tipo de geometria — Retorna o tipo de geometria de valor ST_Geometry.

Synopsis

texto **GeometryType**(geometria geomA);

Descrição

Retorna o tipo de geometria como uma string. Exemplos: 'LINESTRING', 'POLÍGONO', 'MULTIPOINT', etc.

OGC SPEC s2.1.1.1 - Retorna o nome do sub tipo ocasional da geometria da qual essa geometria ocasiona é um membro. O nome do sub tipo ocasional retorna como uma string.

Note
Essa função também indica se a geometria é medida, retornando uma string da forma 'POINTM'.

Melhorias: 2.0.0 suporte para superfícies poliédricas, triângulos e TIN introduzido.

 This method implements the OGC Simple Features Implementation Specification for SQL 1.1.

 This method supports Circular Strings and Curves.

 This function supports 3d and will not drop the z-index.

 This function supports Polyhedral surfaces.

 This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

Exemplos

```
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
```

```
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 geom
    ) AS g;
 result
--------
 TIN
```

Veja também

ST_GeometryType

7.4.2 ST_Boundary

ST_Boundary — Retorna o encerramento da borda combinatória dessa geometria.

Synopsis

geometria **ST_Boundary**(geometria geomA);

Descrição

Retorna o encerramento do limite combinatório dessa geometria. O limite combinatório é definido com descrito na seção 3.12.3.2 do OGC SPEC. Porque o resultado dessa função é um encerramento, e por isso topologicamente fechado, o limite resultante pode ser representado usando geometrias primitivas representacionais como foi discutido no OGC SPEC, seção 3.12.2.

Desempenhado pelo módulo GEOS

Note
Anterior a 2.0.0, essa função abre uma exceção se usada com `GEOMETRYCOLLECTION`. A partir do 2.0.0 ela vai retornar NULA (entrada não suportada).

This method implements the OGC Simple Features Implementation Specification for SQL 1.1. OGC SPEC s2.1.1.1

This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1.17

This function supports 3d and will not drop the z-index.

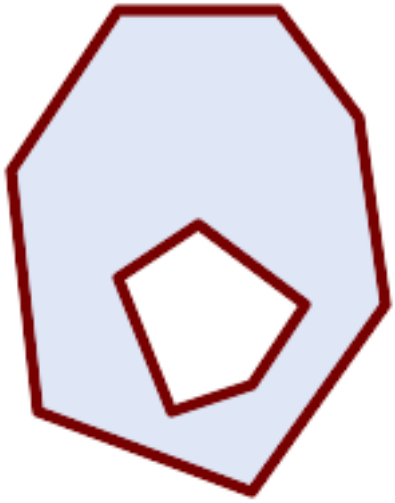
Melhorias: 2.1.0 suporte para Triângulo foi introduzido

Changed: 3.2.0 support for TIN, does not use geos, does not linearize curves

Exemplos

Linestring com pontos de limite cobertos

```
SELECT ST_Boundary(geom)
FROM (SELECT 'LINESTRING(100 150,50 60,  ↩
     70 80, 160 170)'::geometry As geom) As f;

ST_AsText output

MULTIPOINT((100 150),(160 170))
```



furos de polígono com multilinestring limite

```
SELECT ST_Boundary(geom)
FROM (SELECT
'POLYGON (( 10 130, 50 190, 110 190, 140  ↩
     150, 150 80, 100 10, 20 40, 10 130 ),
        ( 70 40, 100 50, 120 80, 80 110,  ↩
     50 90, 70 40 ))'::geometry As geom) As f;

ST_AsText output

MULTILINESTRING((10 130,50 190,110  ↩
     190,140 150,150 80,100 10,20 40,10 130),
        (70 40,100 50,120 80,80 110,50  ↩
     90,70 40))
```

```
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))
```

Veja também

ST_AsText, ST_ExteriorRing, ST_MakePolygon

7.4.3 ST_BoundingDiagonal

ST_BoundingDiagonal — Retorna a diagonal da geometria fornecida da caixa limitada.

Synopsis

geometria **ST_BoundingDiagonal**(geometria geom, booleana fits=false);

Descrição

Retorna a diagonal da geometria fornecida da caixa limitada em linestring. Se a entrada da geometria está vazia, a linha diagonal também está, caso contrário é uma linestring de 2-pontos com valores mínimos de cada dimensão no ponto de início e com valores máximos no ponte de fim.

O parâmetro `fits` especifica se o que se encaixa melhor é necessário. Se negativo, a diagonal de uma caixa limitadora de alguma forma pode ser aceita (é mais rápido obter para geometrias com muitos vértices). De qualquer forma, a caixa limitadora da linha diagonal retornada sempre cobre a geometria de entrada.

A linestring da geometria retornada sempre retém SRID e dimensionalidade (Z e M presentes) da geometria de entrada.

Note
Em casos degenerados (um único vértice na entrada) a linestring retornada será topologicamente inválida (sem interior). Isso não não torna o retorno semanticamente inválido.

Disponibilidade: 2.2.0

 This function supports 3d and will not drop the z-index.

 This function supports M coordinates.

Exemplos

```
-- Get the minimum X in a buffer around a point
SELECT ST_X(ST_StartPoint(ST_BoundingDiagonal(
  ST_Buffer(ST_Point(0,0),10)
)));
 st_x
------
  -10
```

Veja também

ST_StartPoint, ST_EndPoint, ST_X, ST_Y, ST_Z, ST_M, &&&

7.4.4 ST_CoordDim

ST_CoordDim — Retorna a dimensão da coordenada do valor ST_Geometry.

Synopsis

inteiro **ST_CoordDim**(geometria geomA);

Descrição

Retorna a dimensão da coordenada do valor ST_Geometry.

Esse é o pseudônimo condescendente do MM para ST_NDims

This method implements the OGC Simple Features Implementation Specification for SQL 1.1.

This method implements the SQL/MM specification. SQL-MM 3: 5.1.3

This method supports Circular Strings and Curves.

This function supports 3d and will not drop the z-index.

This function supports Polyhedral surfaces.

This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

Exemplos

```
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
```

Veja também

ST_NDims

7.4.5 ST_Dimension

ST_Dimension — Retorna a dimensão da coordenada do valor ST_Geometry.

Synopsis

inteiro **ST_Dimension**(geometria g);

Descrição

A dimensão herdada desse objeto geométrico, que deve ser menor que ou igual à dimensão coordenada. OGC SPEC s2.1.1.1 - retorna 0 para `PONTO`, 1 para `LINESTRING`, 2 para `POLÍGONO`, e a dimensão mais larga dos componentes de uma `COLEÇÃODEGEOMETR`
Se desconhecida (geometria vazia) nula é retornada.

This method implements the SQL/MM specification. SQL-MM 3: 5.1.2

Melhorias: 2.0.0 suporte para superfícies poliédricas e TINs foi introduzido. Não abre mais exceção se uma geometria vazia é dada.

Note
Anterior à 2.0.0, essa função abre uma exceção se usada com uma geometria vazia.

This function supports Polyhedral surfaces.

This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

Exemplos

```
SELECT ST_Dimension('GEOMETRYCOLLECTION(LINESTRING(1 1,0 0),POINT(0 0))');
ST_Dimension
------------
1
```

Veja também

ST_NDims

7.4.6 ST_Dump

ST_Dump — Returns a set of `geometry_dump` rows for the components of a geometry.

Synopsis

geometria **ST_Envelope**(geometria g1);

Descrição

A set-returning function (SRF) that extracts the components of a geometry. It returns a set of geometry_dump rows, each containing a geometry (*geom* field) and an array of integers (*path* field).

For an atomic geometry type (POINT,LINESTRING,POLYGON) a single record is returned with an empty *path* array and the input geometry as *geom*. For a collection or multi-geometry a record is returned for each of the collection components, and the *path* denotes the position of the component inside the collection.

ST_Dump is useful for expanding geometries. It is the inverse of a ST_GeomCollFromText / GROUP BY, in that it creates new rows. For example it can be use to expand MULTIPOLYGONS into POLYGONS.

Melhorias: 2.0.0 suporte para superfícies poliédricas, triângulos e TIN introduzido.

Availability: PostGIS 1.0.0RC1. Requires PostgreSQL 7.3 or higher.

Note
Anteriores a 1.3.4, essa função falha se usada com geometrias que contêm CURVAS. Isso é consertado em 1.3.4+

 This method supports Circular Strings and Curves.

 This function supports Polyhedral surfaces.

 This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

 This function supports 3d and will not drop the z-index.

Exemplos Padrão

```
SELECT sometable.field1, sometable.field1,
      (ST_Dump(sometable.geom)).geom AS 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)
```

Exemplos de Superfícies Poliédricas, TIN e Triângulos

```
-- 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 ↩
    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;

 path |                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))
```

Veja também

geometry_dump, ST_GeomFromEWKT, ST_Dump, ST_GeometryN, ST_NumGeometries

7.4.7 ST_NumPoints

ST_NumPoints — Retorna um texto resumo dos conteúdos da geometria.

Synopsis

geometria **ST_Points**(geometria geom);

Descrição

A set-returning function (SRF) that extracts the coordinates (vertices) of a geometry. It returns a set of geometry_dump rows, each containing a geometry (*geom* field) and an array of integers (*path* field).

- the *geom* field `POINT`s represent the coordinates of the supplied geometry.
- the *path* field (an `integer[]`) is an index enumerating the coordinate positions in the elements of the supplied geometry. The indices are 1-based. For example, for a `LINESTRING` the paths are `{i}` where `i` is the `nth` coordinate in the `LINESTRING`. For a `POLYGON` the paths are `{i, j}` where `i` is the ring number (1 is outer; inner rings follow) and `j` is the coordinate position in the ring.

To obtain a single geometry containing the coordinates use ST_Points.

Enhanced: 2.1.0 Faster speed. Reimplemented as native-C.

Melhorias: 2.0.0 suporte para superfícies poliédricas, triângulos e TIN introduzido.

Disponibilidade: 1.2.2

This method supports Circular Strings and Curves.

This function supports Polyhedral surfaces.

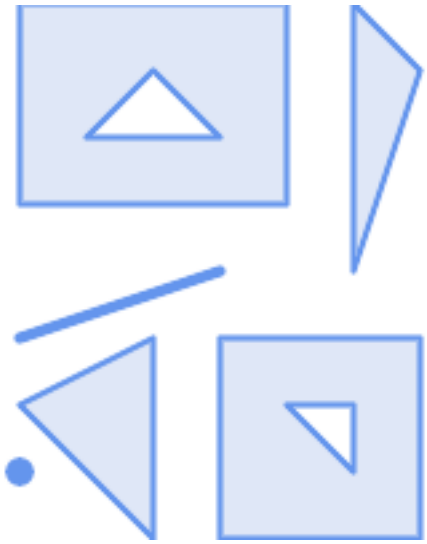
This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

This function supports 3d and will not drop the z-index.

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)
```

Exemplos Padrão



```
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, 2 0 )),
          POLYGON (( 3 0, 3 3, 6 3, 6 0, 3 0 ),
                   ( 5 1, 4 2, 5 2, 5 1 )),
          MULTIPOLYGON (
                  (( 0 5, 0 8, 4 8, 4 5, 0 5 ),
                   ( 1 6, 3 6, 2 7, 1 6 )),
                  (( 5 4, 5 8, 6 7, 5 4 ))
          )
        )'::geometry AS geom
    ) AS g
  ) j;

   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)
```

Exemplos de Superfícies Poliédricas, TIN e Triângulos

```
-- Polyhedral surface cube --
SELECT (g.gdump).path, ST_AsEWKT((g.gdump).geom) as wkt
  FROM
    (SELECT
       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 --
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)
```

Veja também

geometry_dump, ST_GeomFromEWKT, ST_Dump, ST_GeometryN, ST_NumGeometries

7.4.8 ST_NumPoints

ST_NumPoints — Retorna um texto resumo dos conteúdos da geometria.

Synopsis

geometria **ST_Points**(geometria geom);

Descrição

A set-returning function (SRF) that extracts the segments of a geometry. It returns a set of geometry_dump rows, each containing a geometry (*geom* field) and an array of integers (*path* field).

- Retorna VERDADEIRO se essa LINESTRING for fechada e simples.
- the *path* field (an integer[]) is an index enumerating the segment start point positions in the elements of the supplied geometry. The indices are 1-based. For example, for a LINESTRING the paths are {i} where i is the nth segment start point in the LINESTRING. For a POLYGON the paths are {i, j} where i is the ring number (1 is outer; inner rings follow) and j is the segment start point position in the ring.

Disponibilidade: 2.2.0

 This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

 This function supports 3d and will not drop the z-index.

Exemplos Padrão

```
SELECT path, ST_AsText(geom)
FROM (
    SELECT (ST_DumpSegments(g.geom)).*
    FROM (SELECT 'GEOMETRYCOLLECTION(
    LINESTRING(1 1, 3 3, 4 4),
    POLYGON((5 5, 6 6, 7 7, 5 5))
)'::geometry AS geom
        ) AS g
) j;

  path   &#x2502;      st_astext
---------------------------------
 {1,1}   &#x2502; LINESTRING(1 1,3 3)
 {1,2}   &#x2502; LINESTRING(3 3,4 4)
 {2,1,1} &#x2502; LINESTRING(5 5,6 6)
 {2,1,2} &#x2502; LINESTRING(6 6,7 7)
 {2,1,3} &#x2502; LINESTRING(7 7,5 5)
(5 rows)
```

Exemplos de Superfícies Poliédricas, TIN e Triângulos

```
-- Triangle --
SELECT path, ST_AsText(geom)
FROM (
    SELECT (ST_DumpSegments(g.geom)).*
    FROM (SELECT 'TRIANGLE((
        0 0,
        0 9,
        9 0,
        0 0
    ))'::geometry AS geom
        ) AS g
) j;

 path  &#x2502;      st_astext
 ---------------------------------
 {1,1} &#x2502; LINESTRING(0 0,0 9)
 {1,2} &#x2502; LINESTRING(0 9,9 0)
 {1,3} &#x2502; LINESTRING(9 0,0 0)
(3 rows)
```

```
-- TIN --
SELECT path, ST_AsEWKT(geom)
FROM (
    SELECT (ST_DumpSegments(g.geom)).*
    FROM (SELECT '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
    ))
    )'::geometry AS geom
```

```
        ) AS g
) j;

  path   &#x2502;        st_asewkt
  --------------------------------
 {1,1,1} &#x2502; LINESTRING(0 0 0,0 0 1)
 {1,1,2} &#x2502; LINESTRING(0 0 1,0 1 0)
 {1,1,3} &#x2502; LINESTRING(0 1 0,0 0 0)
 {2,1,1} &#x2502; LINESTRING(0 0 0,0 1 0)
 {2,1,2} &#x2502; LINESTRING(0 1 0,1 1 0)
 {2,1,3} &#x2502; LINESTRING(1 1 0,0 0 0)
(6 rows)
```

Veja também

geometry_dump, ST_GeomCollFromText, ST_Dump, ST_NumInteriorRing,

7.4.9 ST_NRings

ST_NRings — Returns a set of `geometry_dump` rows for the exterior and interior rings of a Polygon.

Synopsis

geometria **ST_ExteriorRing**(geometry a_polygon);

Descrição

A set-returning function (SRF) that extracts the rings of a polygon. It returns a set of geometry_dump rows, each containing a geometry (*geom* field) and an array of integers (*path* field).

The *geom* field contains each ring as a POLYGON. The *path* field is an integer array of length 1 containing the polygon ring index. The exterior ring (shell) has index 0. The interior rings (holes) have indices of 1 and higher.

Note
Isso não funcionará para MULTIPOLÍGONOS. Use em conjunção com ST_Dump para MULTIPOLÍGONOS.

Availability: PostGIS 1.1.3. Requires PostgreSQL 7.3 or higher.

 This function supports 3d and will not drop the z-index.

Exemplos

General form of query.

```
SELECT polyTable.field1, polyTable.field1,
          (ST_DumpRings(polyTable.geom)).geom As geom
FROM polyTable;
```

A polygon with a single hole.

```
SELECT path, ST_AsEWKT(geom) As geom
        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 |                                              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))
```

Veja também

geometry_dump, ST_GeomFromEWKT, ST_Dump, ST_GeometryN, ST_NumGeometries

7.4.10 ST_EndPoint

ST_EndPoint — Retorna o número de pontos em um valor ST_LineString ou ST_CircularString.

Synopsis

geometria **ST_Points**(geometria geom);

Descrição

Retorna ao último ponto de uma `LINESTRING` ou `CIRCULARLINESTRING` geometria como um `PONTO` ou `NULO` se o parâmetro de entrada não é uma `LINESTRING`.

This method implements the SQL/MM specification. SQL-MM 3: 7.1.4

This function supports 3d and will not drop the z-index.

This method supports Circular Strings and Curves.

Note

Alterações: 2.0.0 não funciona mais com geometrias de multilinestrings. Em verões mais antigas do PostGIS -- uma linha multilinestring sozinha trabalharia normalmente com essa função e voltaria o ponto de início. Na 2.0.0 ela retorna NULA como qualquer outra multilinestring. O antigo comportamento não foi uma característica documentada, mas as pessoas que consideravam que tinham seus dados armazenados como uma LINESTRING, agora podem experimentar essas que retornam NULAS em 2.0.

Exemplos

End point of a LineString

```
postgis=# SELECT ST_AsText(ST_EndPoint('LINESTRING(1 1, 2 2, 3 3)'::geometry));
 st_astext
------------
 POINT(3 3)
```

End point of a non-LineString is NULL

```
SELECT ST_EndPoint('POINT(1 1)'::geometry) IS NULL AS is_null;
  is_null
----------
 t
```

End point of a 3D LineString

```
--3d endpoint
SELECT ST_AsEWKT(ST_EndPoint('LINESTRING(1 1 2, 1 2 3, 0 0 5)'));
  st_asewkt
--------------
 POINT(0 0 5)
```

Retorna o número de pontos em um valor ST_LineString ou ST_CircularString.

```
SELECT ST_AsText(ST_EndPoint('CIRCULARSTRING(5 2,-3 1.999999, -2 1, -4 2, 6 3)'::geometry)) ↩
    ;
 st_astext
------------
 POINT(6 3)
```

Veja também

ST_PointN, ST_StartPoint

7.4.11 ST_Envelope

ST_Envelope — Retorna uma geometria representando a precisão da dobrada (float8) da caixa limitada da geometria fornecida.

Synopsis

geometria **ST_Envelope**(geometria g1);

Descrição

Retorna o limite mínimo da caixa float8 para a geometria fornecida, com uma geometria. O polígono é definido pelos pontos de canto da caixa limitada ((MINX, MINY), (MINX, MAXY), (MAXX, MAXY), (MAXX, MINY), (MINX, MINY)). (PostGIS irá adicionar uma ZMIN/ZMAX coordenada também).

Casos degenerados (linhas verticais, pontos) irão retornar como uma geometria de dimensão menor que POLÍGONO, ie. PONTO ou LINESTRING.

Disponibilidade: 1.5.0 comportamento alterado para saída de precisão dupla ao invés de float4

This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.1

This method implements the SQL/MM specification. SQL-MM 3: 5.1.19

Exemplos

```
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;
```

Envelope of a point and linestring.

```
SELECT ST_AsText(ST_Envelope(
                ST_Collect(
                        ST_GeomFromText('LINESTRING(55 75,125 150)'),
                                ST_Point(20, 80))
```

```
                                    )) As wktenv;
wktenv
-----------
POLYGON((20 75,20 150,125 150,125 75,20 75))
```

Veja também

Box2D, Box3D, ST_OrientedEnvelope

7.4.12 ST_ExteriorRing

ST_ExteriorRing — Retorna o número de anéis interiores de um polígono.

Synopsis

geometria **ST_ExteriorRing**(geometry a_polygon);

Descrição

Retorna uma line string representando o anel exterior da geometria `POLÍGONO`. Retorna NULA se a geometria não for um polígono.

Note
Isso não funcionará para MULTIPOLÍGONOS. Use em conjunção com ST_Dump para MULTIPOLÍGONOS.


This method implements the OGC Simple Features Implementation Specification for SQL 1.1. 2.1.5.1


This method implements the SQL/MM specification. SQL-MM 3: 8.2.3, 8.3.3


This function supports 3d and will not drop the z-index.

Exemplos

```
--If you have a table of polygons
SELECT gid, ST_ExteriorRing(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(geom)) AS erings
        FROM (SELECT gid, (ST_Dump(geom)).geom As 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)
```

Veja também

ST_InteriorRingN, ST_Boundary, ST_NumInteriorRings

7.4.13 ST_GeometryN

ST_GeometryN — Retorna o tipo de geometria de valor ST_Geometry.

Synopsis

geometria **ST_GeometryN**(geometria geomA, inteiro n);

Descrição

Retorna a geometria de 1-base Nth se a geometria é uma GEOMETRYCOLLECTION, (MULTI)POINT, (MULTI)LINESTRING, MULTICURVE ou (MULTI)POLYGON, POLYHEDRALSURFACE. Senão, retorna NULA.

Note
O Index é 1-base como para OGC specs desde a versão 0.8.0. Versões anteriores implementaram isso como 0-base.

Note
Se você quiser extrair todas as geometrias, de uma geometria, ST_Dump é mais eficiente e também funcionará para geometrias singulares.

Melhorias: 2.0.0 suporte para superfícies poliédricas, triângulos e TIN introduzido.

Alterações: 2.0.0. Versões anteriores voltariam NULAS para geometrias únicas. Isso foi alterado para volrtar a geometria para o caso ST_GeometryN(..,1).


This method implements the OGC Simple Features Implementation Specification for SQL 1.1.


This method implements the SQL/MM specification. SQL-MM 3: 9.1.5


This function supports 3d and will not drop the z-index.


This method supports Circular Strings and Curves.


This function supports Polyhedral surfaces.


This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

Exemplos Padrão

```
--Extracting a subset of points from a 3d multipoint
SELECT n, ST_AsEWKT(ST_GeometryN(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(geom)
        CROSS JOIN generate_series(1,100) n
WHERE n <= ST_NumGeometries(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(geom, n)
FROM sometable CROSS JOIN generate_series(1,100) n
WHERE n <= ST_NumGeometries(geom);
```

Exemplos de Superfícies Poliédricas, TIN e Triângulos

```
-- 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
            ))
```

```
            )')  AS geom
    ) AS g;
-- result --
                 wkt
-----------------------------------
 TRIANGLE((0 0 0,0 1 0,1 1 0,0 0 0))
```

Veja também

ST_Dump, ST_NumGeometries

7.4.14 ST_GeometryType

ST_GeometryType — Retorna o tipo de geometria de valor ST_Geometry.

Synopsis

texto **ST_GeometryType**(geometria g1);

Descrição

Retorna o tipo da geometria como uma string. EX: 'ST_LineString', 'ST_Polygon','ST_MultiPolygon' etc. Essa função difere de GeometryType(geometria) no caso da string e ST na frente que é retornada, bem como o fato que isso não indicará se a geometria é medida.

Melhorias: 2.0.0 suporte a superfícies poliédricas foi introduzido.

This method implements the SQL/MM specification. SQL-MM 3: 5.1.4

This function supports 3d and will not drop the z-index.

This function supports Polyhedral surfaces.

Exemplos

```
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
```

```
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 geom
    ) AS g;
 result
--------
 ST_Tin
```

Veja também

Tipo de geometria

7.4.15 ST_HasArc

ST_HasArc — Tests if a geometry contains a circular arc

Synopsis

booleana **ST_IsEmpty**(geometria geomA);

Descrição

Retorna verdadeiro se essa geometria é uma coleção vazia, polígono, ponto etc.

Disponibilidade: 1.2.2

 This function supports 3d and will not drop the z-index.

 This method supports Circular Strings and Curves.

Exemplos

```
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
```

Veja também

ST_CurveToLine, ST_PointN

7.4.16 ST_InteriorRingN

ST_InteriorRingN — Retorna o número de anéis interiores de um polígono.

Synopsis

geometria **ST_InteriorRingN**(geometria a_polygon, inteiro n);

Descrição

Retorna o anel linestring Nth interior do polígono. Retorna NULO se a geometria não for um polígono ou o dado N está fora da extensão.

Note
Isso não funcionará para MULTIPOLÍGONOS. Use em conjunção com ST_Dump para MULTIPOLÍGONOS.

 This method implements the OGC Simple Features Implementation Specification for SQL 1.1.

 This method implements the SQL/MM specification. SQL-MM 3: 8.2.6, 8.3.5

 This function supports 3d and will not drop the z-index.

Exemplos

```
SELECT ST_AsText(ST_InteriorRingN(geom, 1)) As geom
FROM (SELECT ST_BuildArea(
                ST_Collect(ST_Buffer(ST_Point(1,2), 20,3),
                        ST_Buffer(ST_Point(1, 2), 10,3))) As geom
                )  as foo;
```

Veja também

ST_ExteriorRing, ST_M, ST_X, ST_Y, ST_ZMax, ST_ZMin

7.4.17 ST_IsClosed

ST_IsClosed — Retorna VERDADEIRO se os pontos de começo e fim da LINESTRING são coincidentes. Para superfície poliédrica está fechada (volumétrica).

Synopsis

booleana **ST_IsClosed**(geometria g);

Descrição

Retorna VERDADEIRO se os pontos de começo e fim da LINESTRING são coincidentes. Para superfícies poliédricas, isso lhe diz se a superfície é territorial (aberta) ou volumétrica (fechada).

This method implements the OGC Simple Features Implementation Specification for SQL 1.1.

This method implements the SQL/MM specification. SQL-MM 3: 7.1.5, 9.3.3

Note
SQL-MM define o resultado do ST_IsClosed(NULO) para ser 0, enquanto o PostGIS retorna NULO.

This function supports 3d and will not drop the z-index.

This method supports Circular Strings and Curves.

Melhorias: 2.0.0 suporte a superfícies poliédricas foi introduzido.

This function supports Polyhedral surfaces.

Exemplos de line string e ponto

```
postgis=# SELECT ST_IsClosed('LINESTRING(0 0, 1 1)'::geometry);
 st_isclosed
-------------
 f
(1 row)

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)
```

Exemplos de Superfície Poliedral

```
-- 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
```

Veja também

ST_IsRing

7.4.18 ST_IsCollection

ST_IsCollection — Retorna verdadeiro se essa geometria é uma coleção vazia, polígono, ponto etc.

Synopsis

booleana **ST_IsCollection**(geometria g);

Descrição

Retorna `VERDADEIRO` se o tipo da geometria do argumento é:

- COLEÇÃO DE GEOMETRIA

- MULTI{PONTO, POLÍGONO, LINESTRING, CURVA, SUPERFÍCIE}
- CURVA COMPOSTA

Note
Essa função analisa o tipo da geometria. Isso significa que vai retornar `VERDADEIRO` nas coleções que são vazias ou que contêm apenas um elemento.

 This function supports 3d and will not drop the z-index.

 This method supports Circular Strings and Curves.

Exemplos

```
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)
```

Veja também

ST_NumGeometries

7.4.19 ST_IsEmpty

ST_IsEmpty — Tests if a geometry is empty.

Synopsis

booleana **ST_IsEmpty**(geometria geomA);

Descrição

Retorna verdadeiro se essa geometria se é vazia. Se verdadeira, ela representa uma coleção vazia, polígono, ponto etc.

Note
SQL-MM define o resultado da ST_IsEmpty(NULA) para ser 0, enquanto o PostGIS retorna NULO.

This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.1

This method implements the SQL/MM specification. SQL-MM 3: 5.1.7

This method supports Circular Strings and Curves.

Warning
Alterações: 2.0.0 Nas versões anteriores do PostGIS ST_GeomFromText('GEOMETRYCOLLECTION(EMPTY)') era permitido. Agora isso é ilegal no PostGIS 2.0.0 para se adequar aos padrões SQL/MM.

Exemplos

```
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
------------
 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)
```

7.4.20 ST_IsPolygonCCW

ST_IsPolygonCCW — Tests if Polygons have exterior rings oriented counter-clockwise and interior rings oriented clockwise.

Synopsis

boolean **ST_IsPolygonCCW** (geometry geom);

Descrição

Returns true if all polygonal components of the input geometry use a counter-clockwise orientation for their exterior ring, and a clockwise direction for all interior rings.

Returns true if the geometry has no polygonal components.

Note
Closed linestrings are not considered polygonal components, so you would still get a true return by passing a single closed linestring no matter its orientation.

Note
If a polygonal geometry does not use reversed orientation for interior rings (i.e., if one or more interior rings are oriented in the same direction as an exterior ring) then both ST_IsPolygonCW and ST_IsPolygonCCW will return false.

Disponibilidade: 2.2.0

 This function supports 3d and will not drop the z-index.

 This function supports M coordinates.

Veja também

ST_ForcePolygonCW , ST_ForcePolygonCCW , ST_IsPolygonCW

7.4.21 ST_IsPolygonCW

ST_IsPolygonCW — Tests if Polygons have exterior rings oriented clockwise and interior rings oriented counter-clockwise.

Synopsis

boolean **ST_IsPolygonCW** (geometry geom);

Descrição

Returns true if all polygonal components of the input geometry use a clockwise orientation for their exterior ring, and a counter-clockwise direction for all interior rings.

Returns true if the geometry has no polygonal components.

Note
Closed linestrings are not considered polygonal components, so you would still get a true return by passing a single closed linestring no matter its orientation.

Note
If a polygonal geometry does not use reversed orientation for interior rings (i.e., if one or more interior rings are oriented in the same direction as an exterior ring) then both ST_IsPolygonCW and ST_IsPolygonCCW will return false.

Disponibilidade: 2.2.0

This function supports 3d and will not drop the z-index.

This function supports M coordinates.

Veja também

ST_ForcePolygonCW , ST_ForcePolygonCCW , ST_IsPolygonCW

7.4.22 ST_IsRing

ST_IsRing — Tests if a LineString is closed and simple.

Synopsis

booleana **ST_IsRing**(geometria g);

Descrição

Retorna `VERDADEIRO` se essa `LINESTRING` for ST_IsClosed (`ST_StartPoint` ((*g*)) ~= `ST_Endpoint` ((*g*))) e ST_IsSimple (não cruzar consigo mesma).

This method implements the OGC Simple Features Implementation Specification for SQL 1.1. 2.1.5.1

This method implements the SQL/MM specification. SQL-MM 3: 7.1.6

Note
SQL-MM define o resultado do `ST_IsRing(NULO)` para ser 0, enquanto o PostGIS retorna `NULO`.

Exemplos

```
SELECT ST_IsRing(geom), ST_IsClosed(geom), ST_IsSimple(geom)
FROM (SELECT 'LINESTRING(0 0, 0 1, 1 1, 1 0, 0 0)'::geometry AS geom) AS foo;
 st_isring | st_isclosed | st_issimple
-----------+-------------+-------------
 t         | t           | t
(1 row)
```

```
SELECT ST_IsRing(geom), ST_IsClosed(geom), ST_IsSimple(geom)
FROM (SELECT 'LINESTRING(0 0, 0 1, 1 0, 1 1, 0 0)'::geometry AS geom) AS foo;
 st_isring | st_isclosed | st_issimple
-----------+-------------+-------------
 f         | t           | f
(1 row)
```

Veja também

ST_IsClosed, ST_IsSimple, ST_StartPoint, ST_EndPoint

7.4.23 ST_IsSimple

ST_IsSimple — Retorna (VERDADEIRA) se essa geometria não tem nenhum ponto irregular, como auto intersecção ou tangenciação.

Synopsis

booleana **ST_IsSimple**(geometria geomA);

Descrição

Retorna verdadeira se essa geometria não tem nenhum ponto geométrico irregular, como auto intersecção ou tangenciação. Para maiores informações na definição OGC da simplicidade e validade das geometrias, use "Ensuring OpenGIS compliancy of geometries"

Note
SQL-MM define o resultado da ST_IsSimple(NULA) para ser 0, enquanto o PostGIS retorna NULO.


This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.1


This method implements the SQL/MM specification. SQL-MM 3: 5.1.8


This function supports 3d and will not drop the z-index.

Exemplos

```
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)
```

Veja também

ST_IsValid

7.4.24 ST_M

ST_M — Returns the M coordinate of a Point.

Synopsis

float **ST_M**(geometria a_point);

Descrição

Retorna a coordenada M do ponto, ou NULA se não estiver disponível. Entrada deve ser um ponto.

Note
Isso não faz parte (ainda) do OGC spec, mas está listado aqui para completar a função lista do ponto coordenado extrator.

 This method implements the OGC Simple Features Implementation Specification for SQL 1.1.

 This method implements the SQL/MM specification.

 This function supports 3d and will not drop the z-index.

Exemplos

```
SELECT ST_M(ST_GeomFromEWKT('POINT(1 2 3 4)'));
 st_m
------
        4
(1 row)
```

Veja também

ST_GeomFromEWKT, ST_X, ST_Y, ST_Z

7.4.25 ST_MemSize

ST_MemSize — Retorna o tipo de geometria de valor ST_Geometry.

Synopsis

inteiro **ST_NRings**(geometria geomA);

Descrição

Retorna o tipo de geometria de valor ST_Geometry.

This complements the PostgreSQL built-in database object functions pg_column_size, pg_size_pretty, pg_relation_size, pg_total_relatio

Note

pg_relation_size which gives the byte size of a table may return byte size lower than ST_MemSize. This is because pg_relation_size does not add toasted table contribution and large geometries are stored in TOAST tables.
pg_total_relation_size - includes, the table, the toasted tables, and the indexes.
pg_column_size returns how much space a geometry would take in a column considering compression, so may be lower than ST_MemSize

 This function supports 3d and will not drop the z-index.

 This method supports Circular Strings and Curves.

 This function supports Polyhedral surfaces.

 This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

Changed: 2.2.0 name changed to ST_MemSize to follow naming convention.

Exemplos

```
--Return how much byte space Boston takes up  in our Mass data set
SELECT pg_size_pretty(SUM(ST_MemSize(geom))) as totgeomsum,
pg_size_pretty(SUM(CASE WHEN town = 'BOSTON' THEN ST_MemSize(geom) ELSE 0 END)) As bossum,
CAST(SUM(CASE WHEN town = 'BOSTON' THEN ST_MemSize(geom) ELSE 0 END)*1.00 /
                SUM(ST_MemSize(geom))*100 As numeric(10,2)) As perbos
FROM towns;

totgeomsum        bossum        perbos
----------        ------        ------
1522 kB                30 kB        1.99


SELECT ST_MemSize(ST_GeomFromText('CIRCULARSTRING(220268 150415,220227 150505,220227  ↩
    150406)'));

---
73

--What percentage of our table is taken up by just the geometry
SELECT pg_total_relation_size('public.neighborhoods') As fulltable_size, sum(ST_MemSize( ↩
    geom)) As geomsize,
sum(ST_MemSize(geom))*1.00/pg_total_relation_size('public.neighborhoods')*100 As pergeom
FROM neighborhoods;
fulltable_size geomsize  pergeom
-----------------------------------------------
262144         96238         36.71188354492187500000
```

7.4.26 ST_NDims

ST_NDims — Retorna a dimensão da coordenada do valor ST_Geometry.

Synopsis

integer **ST_NDims**(geometria g1);

Descrição

Retorna a dimensão coordenada da geometria. O PostGIS suporta 2 - (x,y) , 3 - (x,y,z) ou 2D com medida - x,y,m, e 4 - 3D com espaço de medida x,y,z,m

 This function supports 3d and will not drop the z-index.

Exemplos

```
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
```

Veja também

ST_CoordDim, ST_Dimension, ST_GeomFromEWKT

7.4.27 ST_NPoints

ST_NPoints — Retorna o número de pontos (vértices) em uma geometria.

Synopsis

inteiro **ST_NPoints**(geometria g1);

Descrição

Retorna o número de pontos em uma geometria. Funciona para todas as geometrias.

Melhorias: 2.0.0 suporte a superfícies poliédricas foi introduzido.

Note

Anteriores a 1.3.4, essa função falha se usada com geometrias que contêm CURVAS. Isso é consertado em 1.3.4+

 This function supports 3d and will not drop the z-index.

 This method supports Circular Strings and Curves.

 This function supports Polyhedral surfaces.

Exemplos

```
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
```

Veja também

ST_NumPoints

7.4.28 ST_NRings

ST_NRings — Retorna o número de anéis interiores de um polígono.

Synopsis

inteiro **ST_NRings**(geometria geomA);

Descrição

Se a geometria for um polígono ou multi polígono, retorna o número de anéis. Diferente do NumInteriorRings, esse conta os anéis de fora também.

 This function supports 3d and will not drop the z-index.

 This method supports Circular Strings and Curves.

Exemplos

```
SELECT ST_NRings(geom) As Nrings, ST_NumInteriorRings(geom) As ninterrings
                                        FROM (SELECT ST_GeomFromText('POLYGON((1 2, 3 4, 5  ↩
                                            6, 1 2))') As geom) As foo;
         nrings | ninterrings
--------+-------------
          1 |           0
(1 row)
```

Veja também

ST_NumInteriorRings

7.4.29 ST_NumGeometries

ST_NumGeometries — Retorna o número de pontos em uma geometria. Funciona para todas as geometrias.

Synopsis

inteiro **ST_NumGeometries**(geometria geom);

Descrição

Retorna o número de geometrias. Se a geometria é uma GEOMETRYCOLLECTION (ou MULTI*), retorna o número de geometria, para geometrias únicas retornará 1, senão retorna NULO.

Melhorias: 2.0.0 suporte para superfícies poliédricas, triângulos e TIN introduzido.

Alterações: 2.0.0 Em versões anteriores retornaria NULO se a geometria não fosse do tipo coleção/MULTI. 2.0.0+ agora retorna 1 para geometrias únicas ex: POLÍGONO, LINESTRING, PONTO.

This method implements the SQL/MM specification. SQL-MM 3: 9.1.4

This function supports 3d and will not drop the z-index.

This function supports Polyhedral surfaces.

This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

Exemplos

```
--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
```

Veja também

ST_GeometryN, ST_Multi

7.4.30 ST_NumInteriorRings

ST_NumInteriorRings — Retorna o número de anéis interiores de um polígono.

Synopsis

inteiro **ST_NumInteriorRings**(geometria a_polygon);

Descrição

Retorna o número de anéis interiores de um polígono. Retorna NULO se a geometria não for um polígono.

This method implements the SQL/MM specification. SQL-MM 3: 8.2.5

Alterações: 2.0.0 - nas versões anteriores isso permitiria um MULTIPOLÍGONO, retornando o número de anéis interiores do primeiro POLÍGONO.

Exemplos

```
--If you have a regular polygon
SELECT gid, field1, field2, ST_NumInteriorRings(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(geom)) AS numholes
FROM (SELECT gid, field1, field2, (ST_Dump(geom)).geom As geom
        FROM sometable) As foo
GROUP BY gid, field1,field2;
```

Veja também

ST_NumInteriorRing, ST_PointN

7.4.31 ST_NumInteriorRing

ST_NumInteriorRing — Retorna o número de anéis interiores de um polígono na geometria. Sinônimo para ST_NumInteriorRings.

Synopsis

inteiro **ST_NumInteriorRing**(geometria a_polygon);

Veja também

ST_NumInteriorRings, ST_PointN

7.4.32 ST_NumPatches

ST_NumPatches — Retorna o número de faces em uma superfícies poliédrica. Retornará nulo para geometrias não poliédricas.

Synopsis

inteiro **ST_NumPatches**(geometria g1);

Descrição

Retorna o número de faces em uma superfície poliédrica. Retornará nulo para geometrias não poliédricas. Isso é um heterônimo para ST_NumGeometries para suportar a nomeação MM. É mais rápido utilizar ST_NumGeometries se você não se importa com a convenção MM.

Disponibilidade: 2.0.0

This function supports 3d and will not drop the z-index.

This method implements the OGC Simple Features Implementation Specification for SQL 1.1.

This method implements the SQL/MM specification. SQL-MM ISO/IEC 13249-3: 8.5

This function supports Polyhedral surfaces.

Exemplos

```
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
```

Veja também

ST_GeomFromEWKT, ST_NumGeometries

7.4.33 ST_NumPoints

ST_NumPoints — Retorna o número de pontos em um valor ST_LineString ou ST_CircularString.

Synopsis

inteiro **ST_NumPoints**(geometria g1);

Descrição

Retorna o número de pontos em um valor ST_LineString ou ST_CircularString. Anteriores a 1.4 só funcionam com Linestrings como as specs declaram. A partir de 1.4 isso é um heterônimo para ST_NPoints, que retorna o número de vértices apenas para as line strings. Considere utilizar ST_NPoints que tem vários objetivos e funciona com vários tipos de geometrias.

 This method implements the OGC Simple Features Implementation Specification for SQL 1.1.

 This method implements the SQL/MM specification. SQL-MM 3: 7.2.4

Exemplos

```
SELECT ST_NumPoints(ST_GeomFromText('LINESTRING(77.29 29.07,77.42 29.26,77.27 29.31,77.29 ↩
    29.07)'));
                --result
                4
```

Veja também

ST_NPoints

7.4.34 ST_PatchN

ST_PatchN — Retorna o tipo de geometria de valor ST_Geometry.

Synopsis

geometria **ST_PatchN**(geometria geomA, inteiro n);

Descrição

> Retorna a geometria (face) de 1-base Nth se a geometria é POLYHEDRALSURFACE, POLYHEDRALSURFACEM. Senão, retorna NULA. Retorna a mesma resposta como ST_GeometryN para superfícies poliédricas. Utilizar ST_GemoetryN é mais rápido.

Note
Index é 1-base.

Note
Se você quiser extrair todas as geometrias, de uma geometria, ST_Dump é mais eficiente.

Disponibilidade: 2.0.0

 This method implements the SQL/MM specification. SQL-MM ISO/IEC 13249-3: 8.5

 This function supports 3d and will not drop the z-index.

 This function supports Polyhedral surfaces.

Exemplos

```
--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))
```

Veja também

ST_AsEWKT, ST_GeomFromEWKT, ST_Dump, ST_GeometryN, ST_NumGeometries

7.4.35 ST_PointN

ST_PointN — Retorna o número de pontos em um valor ST_LineString ou ST_CircularString.

Synopsis

geometria **ST_PointN**(geometria a_linestring, inteiro n);

Descrição

Retorna o ponto Nth na primeira linestring ou linestring circular na geometria. Valores negativos são contados tardiamente do fim da linestring, tornando o ponto -1 o último ponto. Retorna NULA se não há uma linestring na geometria.

Note
O Index é 1-base como para OGC specs desde a versão 0.8.0. Indexing atrasado (negativo) não está nas versões OGC anteriores implementadas com 0-base.

Note
Se você quiser o ponto nth de cada line string em uma multilinestring, utilize em conjunção com ST_Dump


This method implements the OGC Simple Features Implementation Specification for SQL 1.1.


This method implements the SQL/MM specification. SQL-MM 3: 7.2.5, 7.3.5


This function supports 3d and will not drop the z-index.


This method supports Circular Strings and Curves.

Note
Alterações: 2.0.0 não funciona mais com geometrias multilinestrings únicas. Em verões mais antigas do PostGIS -- uma única linha multilinestring trabalharia normalmente e retornaria o ponto inicial. Na 2.0.0 só retorna NULA como qualquer outra multilinestring.
Alterações: 2.3.0 : indexing negativo disponível (-1 é o último ponto)

Exemplos

```
-- 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)
(1 row)
```

```
SELECT ST_AsText(f)
FROM ST_GeomFromText('LINESTRING(0 0 0, 1 1 1, 2 2 2)') AS g
  ,ST_PointN(g, -2) AS f; -- 1 based index

    st_astext
-----------------
 POINT Z (1 1 1)
(1 row)
```

Veja também

ST_NPoints

7.4.36 ST_Points

ST_Points — Retorna uma multilinestring contendo todas as coordenadas de uma geometria.

Synopsis

geometria **ST_Points**(geometria geom);

Descrição

Returns a MultiPoint containing all the coordinates of a geometry. Duplicate points are preserved, including the start and end points of ring geometries. (If desired, duplicate points can be removed by calling ST_RemoveRepeatedPoints on the result).

To obtain information about the position of each coordinate in the parent geometry use ST_NumPoints.

M and Z coordinates are preserved if present.

 This method supports Circular Strings and Curves.

 This function supports 3d and will not drop the z-index.

Disponibilidade: 2.3.0

Exemplos

```
SELECT ST_AsText(ST_Points('POLYGON Z ((30 10 4,10 30 5,40 40 6, 30 10))'));

--result
MULTIPOINT Z ((30 10 4),(10 30 5),(40 40 6),(30 10 4))
```

Veja também

ST_RemoveRepeatedPoints, ST_PointN

7.4.37 ST_StartPoint

ST_StartPoint — Returns the first point of a LineString.

Synopsis

geometria **ST_StartPoint**(geometria geomA);

Descrição

Retorna ao último ponto de uma `LINESTRING` ou `CIRCULARLINESTRING` geometria como um `PONTO` ou `NULO` se o parâmetro de entrada não é uma `LINESTRING`.

This method implements the SQL/MM specification. SQL-MM 3: 7.1.3

This function supports 3d and will not drop the z-index.

This method supports Circular Strings and Curves.

Note

Enhanced: 3.2.0 returns a point for all geometries. Prior behavior returns NULLs if input was not a LineString.
Alterações: 2.0.0 não funciona mais com geometrias de multilinestrings. Em verões mais antigas do PostGIS -- uma linha multilinestring sozinha trabalharia normalmente com essa função e voltaria o ponto de início. Na 2.0.0 ela retorna NULA como qualquer outra multilinestring. O antigo comportamento não foi uma característica documentada, mas as pessoas que consideravam que tinham seus dados armazenados como uma LINESTRING, agora podem experimentar essas que retornam NULAS em 2.0.

Exemplos

Start point of a LineString

```
SELECT ST_AsText(ST_StartPoint('LINESTRING(0 1, 0 2)'::geometry));
 st_astext
------------
 POINT(0 1)
```

Start point of a non-LineString is NULL

```
SELECT ST_StartPoint('POINT(0 1)'::geometry) IS NULL AS is_null;
  is_null
----------
 t
```

Start point of a 3D LineString

```
SELECT ST_AsEWKT(ST_StartPoint('LINESTRING(0 1 1, 0 2 2)'::geometry));
 st_asewkt
------------
 POINT(0 1 1)
```

Retorna o número de pontos em um valor ST_LineString ou ST_CircularString.

```
SELECT ST_AsText(ST_StartPoint('CIRCULARSTRING(5 2,-3 1.999999, -2 1, -4 2, 6 3)'::geometry ↩
    ));
 st_astext
------------
 POINT(5 2)
```

Veja também

ST_EndPoint, ST_PointN

7.4.38 ST_Summary

ST_Summary — Retorna um texto resumo dos conteúdos da geometria.

Synopsis

text **ST_Summary**(geometry g);
text **ST_Summary**(geography g);

Descrição

Retorna um texto resumo dos conteúdos da geometria.

As bandeiras mostraram colchetes depois do tipo de geometria ter o seguinte significado:

- M: tem ordenada M
- Z: tem ordenada Z
- B: tem uma caixa limitante salva
- G: é geodésico (geografia)
- S: tem um sistema de referência espacial

 This method supports Circular Strings and Curves.

 This function supports Polyhedral surfaces.

 This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

Disponibilidade: 1.2.2

Melhorias: 2.0.0 suporte para geografia adicionado

melhorias: 2.1.0 Bandeira S para indicar se existe um sistema de referência espacial conhecido

Melhorias: 2.2.0 Suporte para TIN e Curvas adicionado

Exemplos

```
=# 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)


=# 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)
```

Veja também

PostGIS_DropBBox, PostGIS_AddBBox, ST_Force3DM, ST_Force3DZ, ST_Force2D, geografia

ST_IsValid, ST_IsValid, ST_IsValidReason, ST_IsValidDetail

7.4.39 ST_X

ST_X — Returns the X coordinate of a Point.

Synopsis

float **ST_X**(geometria a_point);

Descrição

Retorna a coordenada X do ponto, ou NULA se não estiver disponível. Entrada deve ser um ponto.

Note
To get the minimum and maximum X value of geometry coordinates use the functions ST_XMin and ST_XMax.


This method implements the SQL/MM specification. SQL-MM 3: 6.1.3


This function supports 3d and will not drop the z-index.

Exemplos

```
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)
```

Veja também

ST_Centroid, ST_GeomFromEWKT, ST_M, ST_XMax, ST_XMin, ST_Y, ST_Z

7.4.40 ST_Y

ST_Y — Returns the Y coordinate of a Point.

Synopsis

float **ST_Y**(geometria a_point);

Descrição

Retorna a coordenada Y do ponto, ou NULA se não estiver disponível. Entrada deve ser um ponto.

Note
To get the minimum and maximum Y value of geometry coordinates use the functions ST_YMin and ST_YMax.

This method implements the OGC Simple Features Implementation Specification for SQL 1.1.

This method implements the SQL/MM specification. SQL-MM 3: 6.1.4

This function supports 3d and will not drop the z-index.

Exemplos

```
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)
```

Veja também

ST_Centroid, ST_GeomFromEWKT, ST_M, ST_X, ST_YMax, ST_YMin, ST_Z

7.4.41 ST_Z

ST_Z — Returns the Z coordinate of a Point.

Synopsis

float **ST_Z**(geometria a_point);

Descrição

Retorna a coordenada Z do ponto, ou NULA se não estiver disponível. Entrada deve ser um ponto.

Note
To get the minimum and maximum Z value of geometry coordinates use the functions ST_ZMin and ST_ZMax.

 This method implements the SQL/MM specification.

 This function supports 3d and will not drop the z-index.

Exemplos

```
SELECT ST_Z(ST_GeomFromEWKT('POINT(1 2 3 4)'));
 st_z
------
        3
(1 row)
```

Veja também

ST_GeomFromEWKT, ST_M, ST_X, ST_Y, ST_ZMax, ST_ZMin

7.4.42 ST_Zmflag

ST_Zmflag — Retorna a dimensão da coordenada do valor ST_Geometry.

Synopsis

smallint **ST_Zmflag**(geometria geomA);

Descrição

Retorna a dimensão da coordenada do valor ST_Geometry.

Values are: 0 = 2D, 1 = 3D-M, 2 = 3D-Z, 3 = 4D.

 This function supports 3d and will not drop the z-index.

 This method supports Circular Strings and Curves.

Exemplos

```
SELECT ST_Zmflag(ST_GeomFromEWKT('LINESTRING(1 2, 3 4)'));
 st_zmflag
-----------
                0

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
```

Veja também

ST_CoordDim, ST_NDims, ST_Dimension

7.5 Editores de geometria

7.5.1 ST_AddPoint

ST_AddPoint — Adicione um ponto para uma LineString.

Synopsis

geometry **ST_AddPoint**(geometry linestring, geometry point);

geometry **ST_AddPoint**(geometry linestring, geometry point, integer position = -1);

Descrição

Adds a point to a LineString before the index *position* (using a 0-based index). If the *position* parameter is omitted or is -1 the point is appended to the end of the LineString.

Disponibilitade: 1.1.0

 This function supports 3d and will not drop the z-index.

Exemplos

Add a point to the end of a 3D line

```
SELECT ST_AsEWKT(ST_AddPoint('LINESTRING(0 0 1, 1 1 1)', ST_MakePoint(1, 2, 3)));

    st_asewkt
    ----------
    LINESTRING(0 0 1,1 1 1,1 2 3)
```

Guarantee all lines in a table are closed by adding the start point of each line to the end of the line only for those that are not closed.

```
UPDATE sometable
SET geom = ST_AddPoint(geom, ST_StartPoint(geom))
FROM sometable
WHERE ST_IsClosed(geom) = false;
```

Veja também

ST_RemovePoint, ST_SetPoint

7.5.2 ST_CollectionExtract

ST_CollectionExtract — Given a geometry collection, returns a multi-geometry containing only elements of a specified type.

Synopsis

geometry **ST_CollectionExtract**(geometry collection);

geometry **ST_CollectionExtract**(geometry collection, integer type);

Descrição

Given a geometry collection, returns a homogeneous multi-geometry.

If the `type` is not specified, returns a multi-geometry containing only geometries of the highest dimension. So polygons are preferred over lines, which are preferred over points.

If the `type` is specified, returns a multi-geometry containing only that type. If there are no sub-geometries of the right type, an EMPTY geometry is returned. Only points, lines and polygons are supported. The type numbers are:

- 1 == POINT
- 2 == LINESTRING
- 3 == POLYGON

For atomic geometry inputs, the geometry is retured unchanged if the input type matches the requested type. Otherwise, the result is an EMPTY geometry of the specified type. If required, these can be converted to multi-geometries using ST_Multi.

Warning
MultiPolygon results are not checked for validity. If the polygon components are adjacent or overlapping the result will be invalid. (For example, this can occur when applying this function to an ST_Split result.) This situation can be checked with ST_IsValid and repaired with ST_MakeValid.

Disponibilidade: 1.5.0

Note
Prior to 1.5.3 this function returned atomic inputs unchanged, no matter type. In 1.5.3 non-matching single geometries returned a NULL result. In 2.0.0 non-matching single geometries return an EMPTY result of the requested type.

Exemplos

Extract highest-dimension type:

```
SELECT ST_AsText(ST_CollectionExtract(
        'GEOMETRYCOLLECTION( POINT(0 0), LINESTRING(1 1, 2 2) )'));
    st_astext
    ---------------
    MULTILINESTRING((1 1, 2 2))
```

Extract points (type 1 == POINT):

```
SELECT ST_AsText(ST_CollectionExtract(
        'GEOMETRYCOLLECTION(GEOMETRYCOLLECTION(POINT(0 0)))',
        1 ));
    st_astext
    ---------------
    MULTIPOINT((0 0))
```

Extract lines (type 2 == LINESTRING):

```
SELECT ST_AsText(ST_CollectionExtract(
        'GEOMETRYCOLLECTION(GEOMETRYCOLLECTION(LINESTRING(0 0, 1 1)),LINESTRING(2 2, 3 3)) ↩
            ',
        2 ));
    st_astext
    ---------------
    MULTILINESTRING((0 0, 1 1), (2 2, 3 3))
```

Veja também

ST_CollectionHomogenize, ST_Multi, ST_IsValid, ST_MakeValid

7.5.3 ST_CollectionHomogenize

ST_CollectionHomogenize — Returns the simplest representation of a geometry collection.

Synopsis

geometry **ST_CollectionHomogenize**(geometry collection);

Descrição

Dada uma coleção geométrica, retorna a representação mais simples de seu conteúdo.

- Homogeneous (uniform) collections are returned as the appropriate multi-geometry.
- Heterogeneous (mixed) collections are flattened into a single GeometryCollection.
- Collections containing a single atomic element are returned as that element.
- Atomic geometries are returned unchanged. If required, these can be converted to a multi-geometry using ST_Multi.

Warning
This function does not ensure that the result is valid. In particular, a collection containing adjacent or overlapping Polygons will create an invalid MultiPolygon. This situation can be checked with ST_IsValid and repaired with ST_MakeValid.

Disponibilidade: 2.0.0

Exemplos

Single-element collection converted to an atomic geometry

```
SELECT ST_AsText(ST_CollectionHomogenize('GEOMETRYCOLLECTION(POINT(0 0))'));

        st_astext
        ------------
        POINT(0 0)
```

Nested single-element collection converted to an atomic geometry:

```
SELECT ST_AsText(ST_CollectionHomogenize('GEOMETRYCOLLECTION(MULTIPOINT((0 0)))'));

        st_astext
        ------------
        POINT(0 0)
```

Collection converted to a multi-geometry:

```
SELECT ST_AsText(ST_CollectionHomogenize('GEOMETRYCOLLECTION(POINT(0 0),POINT(1 1))'));

        st_astext
        ---------------------
        MULTIPOINT((0 0),(1 1))
```

Nested heterogeneous collection flattened to a GeometryCollection:

```
SELECT ST_AsText(ST_CollectionHomogenize('GEOMETRYCOLLECTION(POINT(0 0), GEOMETRYCOLLECTION ↩
    ( LINESTRING(1 1, 2 2)))'));

        st_astext
        ---------------------
        GEOMETRYCOLLECTION(POINT(0 0),LINESTRING(1 1,2 2))
```

Collection of Polygons converted to an (invalid) MultiPolygon:

```
SELECT ST_AsText(ST_CollectionHomogenize('GEOMETRYCOLLECTION (POLYGON ((10 50, 50 50, 50  ↩
    10, 10 10, 10 50)), POLYGON ((90 50, 90 10, 50 10, 50 50, 90 50)))'));

        st_astext
        ---------------------
        MULTIPOLYGON(((10 50,50 50,50 10,10 10,10 50)),((90 50,90 10,50 10,50 50,90 50)))
```

Veja também

ST_CollectionExtract, ST_Multi, ST_IsValid, ST_MakeValid

7.5.4 ST_CurveToLine

ST_CurveToLine — Converts a geometry containing curves to a linear geometry.

Synopsis

geometry **ST_CurveToLine**(geometry curveGeom, float tolerance, integer tolerance_type, integer flags);

Descrição

Converts a CIRCULAR STRING to regular LINESTRING or CURVEPOLYGON to POLYGON or MULTISURFACE to MULTIPOLYGON. Useful for outputting to devices that can't support CIRCULARSTRING geometry types

Converts a given geometry to a linear geometry. Each curved geometry or segment is converted into a linear approximation using the given `tolerance` and options (32 segments per quadrant and no options by default).

The 'tolerance_type' argument determines interpretation of the `tolerance` argument. It can take the following values:

- 0 (default): Tolerance is max segments per quadrant.
- 1: Tolerance is max-deviation of line from curve, in source units.
- 2: Tolerance is max-angle, in radians, between generating radii.

The 'flags' argument is a bitfield. 0 by default. Supported bits are:

- 1: Symmetric (orientation idependent) output.
- 2: Retain angle, avoids reducing angles (segment lengths) when producing symmetric output. Has no effect when Symmetric flag is off.

Availability: 1.3.0

Enhanced: 2.4.0 added support for max-deviation and max-angle tolerance, and for symmetric output.

Enhanced: 3.0.0 implemented a minimum number of segments per linearized arc to prevent topological collapse.

 This method implements the OGC Simple Features Implementation Specification for SQL 1.1.

 This method implements the SQL/MM specification. SQL-MM 3: 7.1.7

 This function supports 3d and will not drop the z-index.

 This method supports Circular Strings and Curves.

Exemplos

```
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,
 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)

-- Ensure approximated line is no further than 20 units away from
-- original curve, and make the result direction-neutral
SELECT ST_AsText(ST_CurveToLine(
 'CIRCULARSTRING(0 0,100 -100,200 0)'::geometry,
    20, -- Tolerance
    1, -- Above is max distance between curve and line
    1  -- Symmetric flag
));
st_astext
-----------------------------------------------------------------------------------------------
 LINESTRING(0 0,50 -86.6025403784438,150 -86.6025403784439,200 -1.1331077795296e-13,200 0)
```

Veja também

ST_LineToCurve

7.5.5 ST_Scroll

ST_Scroll — Change start point of a closed LineString.

Synopsis

geometry **ST_Scroll**(geometry linestring, geometry point);

Descrição

Changes the start/end point of a closed LineString to the given vertex *point*.

Availability: 3.2.0

This function supports 3d and will not drop the z-index.

This function supports M coordinates.

Exemplos

Make e closed line start at its 3rd vertex

```
SELECT ST_AsEWKT(ST_Scroll('SRID=4326;LINESTRING(0 0 0 1, 10 0 2 0, 5 5 4 2,0 0 0 1)', ' ↩
    POINT(5 5 4 2)'));

st_asewkt
----------
SRID=4326;LINESTRING(5 5 4 2,0 0 0 1,10 0 2 0,5 5 4 2)
```

Veja também

ST_Normalize

7.5.6 ST_FlipCoordinates

ST_FlipCoordinates — Returns a version of a geometry with X and Y axis flipped.

Synopsis

geometry **ST_FlipCoordinates**(geometry geom);

Descrição

Returns a version of the given geometry with X and Y axis flipped. Useful for fixing geometries which contain coordinates expressed as latitude/longitude (Y,X).

Disponibilidade: 2.0.0

 This method supports Circular Strings and Curves.

 This function supports 3d and will not drop the z-index.

 This function supports M coordinates.

 This function supports Polyhedral surfaces.

 This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

Exemplo

```
SELECT ST_AsEWKT(ST_FlipCoordinates(GeomFromEWKT('POINT(1 2)')));
 st_asewkt
------------
POINT(2 1)
```

Veja também

ST_SwapOrdinates

7.5.7 ST_Force2D

ST_Force2D — Força a geometria para o modo de 2 dimensões.

Synopsis

geometry **ST_Force2D**(geometry geomA);

Descrição

Força a geometria a possuir apenas duas dimensões, para que todas saídas tenham apenas as coordenadas X e Y. Esta função é útil para forçar geometrias de acordo a norma OGC (a OGC apenas especifica geometrias de duas dimensões).

Melhorias: 2.0.0 suporte a superfícies polihédricas foi introduzido.

Alterado: 2.1.0. Até versão 2.0.x isto era chamado de ST_Force_2D.

This method supports Circular Strings and Curves.

This function supports Polyhedral surfaces.

This function supports 3d and will not drop the z-index.

Exemplos

```
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))
```

Veja também

ST_Force3D

7.5.8 ST_Force3D

ST_Force3D — Força a geometria para um modo XYZ. Este é um apelido para a função ST_Force_3DZ.

Synopsis

geometry **ST_Force3D**(geometry geomA, float Zvalue = 0.0);

Descrição

Forces the geometries into XYZ mode. This is an alias for ST_Force3DZ. If a geometry has no Z component, then a `Zvalue` Z coordinate is tacked on.

Melhorias: 2.0.0 suporte a superfícies polihédricas foi introduzido.

Alterado: 2.1.0. Até versão 2.0.x isto era chamado de ST_Force_3D.

Changed: 3.1.0. Added support for supplying a non-zero Z value.

This function supports Polyhedral surfaces.

This method supports Circular Strings and Curves.

This function supports 3d and will not drop the z-index.

Exemplos

```
--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))
```

Veja também

ST_AsEWKT, ST_Force2D, ST_Force3DM, ST_Force3DZ

7.5.9 ST_Force3DZ

ST_Force3DZ — Força as geometrias para o modo XYZ.

Synopsis

geometry **ST_Force3DZ**(geometry geomA, float Zvalue = 0.0);

Descrição

Forces the geometries into XYZ mode. If a geometry has no Z component, then a `Zvalue` Z coordinate is tacked on.

Melhorias: 2.0.0 suporte a superfícies polihédricas foi introduzido.

Alterado: 2.1.0. Até versão 2.0.x isto era chamado de ST_Force_3DZ.

Changed: 3.1.0. Added support for supplying a non-zero Z value.

This function supports Polyhedral surfaces.

This function supports 3d and will not drop the z-index.

This method supports Circular Strings and Curves.

Exemplos

```
--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))
```

Veja também

ST_AsEWKT, ST_Force2D, ST_Force3DM, ST_Force3D

7.5.10 ST_Force3DM

ST_Force3DM — Força as geometrias para o modo XYM.

Synopsis

geometry **ST_Force3DM**(geometry geomA, float Mvalue = 0.0);

Descrição

Forces the geometries into XYM mode. If a geometry has no M component, then a *Mvalue* M coordinate is tacked on. If it has a Z component, then Z is removed

Alterado: 2.1.0. Até a versão 2.0.x esta função era chamada de ST_Force_3DM.

Changed: 3.1.0. Added support for supplying a non-zero M value.

 This method supports Circular Strings and Curves.

Exemplos

```
--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))
```

Veja também

ST_AsEWKT, ST_Force2D, ST_Force3DM, ST_Force3D, ST_GeomFromEWKT

7.5.11 ST_Force4D

ST_Force4D — Força as geometrias para o modo XYZM.

Synopsis

geometry **ST_Force4D**(geometry geomA, float Zvalue = 0.0, float Mvalue = 0.0);

Descrição

Forces the geometries into XYZM mode. *Zvalue* and *Mvalue* is tacked on for missing Z and M dimensions, respectively.

Alterado: 2.1.0. Até a versão 2.0.x esta função era chamada ST_Force_4D.

Changed: 3.1.0. Added support for supplying non-zero Z and M values.

This function supports 3d and will not drop the z-index.

This method supports Circular Strings and Curves.

Exemplos

```
--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))
```

Veja também

ST_AsEWKT, ST_Force2D, ST_Force3DM, ST_Force3D

7.5.12 ST_ForcePolygonCCW

ST_ForcePolygonCCW — Orients all exterior rings counter-clockwise and all interior rings clockwise.

Synopsis

geometry **ST_ForcePolygonCCW** (geometry geom);

Descrição

Forces (Multi)Polygons to use a counter-clockwise orientation for their exterior ring, and a clockwise orientation for their interior rings. Non-polygonal geometries are returned unchanged.

Availability: 2.4.0

This function supports 3d and will not drop the z-index.

This function supports M coordinates.

Veja também

ST_ForcePolygonCW , ST_IsPolygonCCW , ST_IsPolygonCW

7.5.13 ST_ForceCollection

ST_ForceCollection — Converte a geometria para um GEOMETRYCOLLECTION.

Synopsis

geometry **ST_ForceCollection**(geometry geomA);

Descrição

Converte a geometria em um GEOMETRYCOLLECTION. Isto é útil para simplificar a representação WKB.

Melhorias: 2.0.0 suporte a superfícies polihédricas foi introduzido.

Disponibilidade: 1.2.2, antes da versão 1.3.4 esta função irá reportar um erro com curvas. Resolvido na versão 1.3.4+.

Alterado: 2.1.0. Até a versão 2.0.x esta função era chamada de ST_Force_Collection.

 This function supports Polyhedral surfaces.

 This function supports 3d and will not drop the z-index.

 This method supports Circular Strings and Curves.

Exemplos

```
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))
)
```

Veja também

ST_AsEWKT, ST_Force2D, ST_Force3DM, ST_Force3D, ST_GeomFromEWKT

7.5.14 ST_ForcePolygonCW

ST_ForcePolygonCW — Orients all exterior rings clockwise and all interior rings counter-clockwise.

Synopsis

geometry **ST_ForcePolygonCW** (geometry geom);

Descrição

Forces (Multi)Polygons to use a clockwise orientation for their exterior ring, and a counter-clockwise orientation for their interior rings. Non-polygonal geometries are returned unchanged.

Availability: 2.4.0

 This function supports 3d and will not drop the z-index.

 This function supports M coordinates.

Veja também

ST_ForcePolygonCCW , ST_IsPolygonCCW , ST_IsPolygonCW

7.5.15 ST_ForceSFS

ST_ForceSFS — Força as geometrias a utilizarem os tipos disponíveis na especificação SFS 1.1.

Synopsis

geometry **ST_ForceSFS**(geometry geomA);
geometry **ST_ForceSFS**(geometry geomA, text version);

Descrição

 This function supports Polyhedral surfaces.

 This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

 This method supports Circular Strings and Curves.

 This function supports 3d and will not drop the z-index.

7.5.16 ST_ForceRHR

ST_ForceRHR — Força a orientação dos vértices em um polígono a seguir a regra da mão direita.

Synopsis

geometry **ST_ForceRHR**(geometry g);

Descrição

Forces the orientation of the vertices in a polygon to follow a Right-Hand-Rule, in which the area that is bounded by the polygon is to the right of the boundary. In particular, the exterior ring is orientated in a clockwise direction and the interior rings in a counter-clockwise direction. This function is a synonym for ST_ForcePolygonCW

Note
The above definition of the Right-Hand-Rule conflicts with definitions used in other contexts. To avoid confusion, it is recommended to use ST_ForcePolygonCW.

Melhorias: 2.0.0 suporte a superfícies polihédricas foi introduzido.


This function supports 3d and will not drop the z-index.


This function supports Polyhedral surfaces.

Exemplos

```
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)
```

Veja também

ST_ForcePolygonCCW , ST_ForcePolygonCW , ST_IsPolygonCCW , ST_IsPolygonCW , ST_BuildArea, ST_Polygonize, ST_Reverse

7.5.17 ST_ForceCurve

ST_ForceCurve — Converte para cima uma geometria para seu tipo curvo, se aplicável.

Synopsis

geometry **ST_ForceCurve**(geometry g);

Descrição

Transforma uma geometria em sua representação curva, se aplicável. linhas se transformar em compoundcurves, multi-linhas se transformam em multicurves, polígonos em curvepolygons, multi-polígonos em multisurfaces. Se a entrada já é do tipo curvo, a função retorna a mesma entrada·

Disponibilidade: 2.2.0

 This function supports 3d and will not drop the z-index.

 This method supports Circular Strings and Curves.

Exemplos

```
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))'::geometry
  )
);
                               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)
```

Veja também

ST_LineToCurve

7.5.18 ST_LineToCurve

ST_LineToCurve — Converts a linear geometry to a curved geometry.

Synopsis

geometry **ST_LineToCurve**(geometry geomANoncircular);

Descrição

Converts plain LINESTRING/POLYGON to CIRCULAR STRINGs and Curved Polygons. Note much fewer points are needed to describe the curved equivalent.

Note
If the input LINESTRING/POLYGON is not curved enough to clearly represent a curve, the function will return the same input geometry.

Availability: 1.3.0

 This function supports 3d and will not drop the z-index.

 This method supports Circular Strings and Curves.

Exemplos

```
-- 2D Example
SELECT ST_AsText(ST_LineToCurve(foo.geom)) As curvedastext,ST_AsText(foo.geom) As  ↩
    non_curvedastext
    FROM (SELECT ST_Buffer('POINT(1 3)'::geometry, 3) As 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  ↩
                                                                          0.505591163092366,2.14805029
                                                                           0.228361402466141,
                                                                      |  1.58527096604839  ↩
                                                                          0.0576441587903094,1 ↩
                                                                           0,
                                                                      |  0.414729033951621  ↩
                                                                          0.0576441587903077,-0.148050
                                                                           0.228361402466137,
                                                                      |  -0.666710699058802  ↩
                                                                          0.505591163092361,-1.121320
                                                                           0.878679656440353,
                                                                      |  -1.49440883690763  ↩
                                                                          1.33328930094119,-1.77163859
                                                                           1.85194970290472
                                                                      |  --ETC--  ↩
                                                                          ,3.94235584120969  ↩
                                                                          3.58527096604839,4  ↩
                                                                          3))

--3D example
SELECT ST_AsText(ST_LineToCurve(geom)) As curved, ST_AsText(geom) AS not_curved
FROM (SELECT ST_Translate(ST_Force3D(ST_Boundary(ST_Buffer(ST_Point(1,3), 2,2))),0,0,3) AS  ↩
    geom) AS foo;

                        curved                         |               not_curved
-------------------------------------------------------+-------------------------------------------

 CIRCULARSTRING Z (3 3 3,-1 2.99999999999999 3,3 3 3) | LINESTRING Z (3 3 3,2.4142135623731 ↩
     1.58578643762691 3,1 1 3,
                                                      | -0.414213562373092 1.5857864376269  ↩
                                                          3,-1 2.99999999999999 3,
                                                      | -0.414213562373101 4.41421356237309 ↩
                                                           3,
                                                      | 0.999999999999991 5  ↩
                                                          3,2.41421356237309 4.4142135623731 ↩
                                                           3,3 3 3)
(1 row)
```

Veja também

ST_CurveToLine

7.5.19 ST_Multi

ST_Multi — Restitui a geometria como uma MULTI* geometria.

Synopsis

geometry **ST_Multi**(geometry geom);

Descrição

Returns the geometry as a MULTI* geometry collection. If the geometry is already a collection, it is returned unchanged.

Exemplos

```
SELECT ST_AsText(ST_Multi('POLYGON ((10 30, 30 30, 30 10, 10 10, 10 30))'));
                    st_astext
    ------------------------------------------------
    MULTIPOLYGON(((10 30,30 30,30 10,10 10,10 30)))
```

Veja também

ST_AsText

7.5.20 ST_LineExtend

ST_LineExtend — Returns a line with the last and first segments extended the specified distance(s).

Synopsis

geometry **ST_LineExtend**(geometry line, float distance_forward, float distance_backward=0.0);

Descrição

Returns a line with the last and first segments extended the specified distance(s). Distance of zero carries out no extension. Only non-negative distances are allowed. The first (and last) two distinct points in a line are used to determine the direction of projection, duplicate points are ignored.

Availability: 3.4.0

Example: Projected point at 100,000 meters and bearing 45 degrees

```
SELECT ST_AsText(ST_Project('POINT(0 0)'::geography, 100000, radians(45.0)));
-----------------------------------------
 POINT(0.635231029125537 0.639472334729198)
```

Veja também

ST_LocateAlong, ST_Project

7.5.21 ST_Normalize

ST_Normalize — Retorna a geometria na sua forma canônica.

Synopsis

geometry **ST_Normalize**(geometry geom);

Descrição

Retorna a geometria na sua forma normalizada/canônica. Talvez rearranja vértices em anéis de polígonos, anéis em um polígono, elementos em um complexo de multi-geometria.

Mais usada para teste (comparando resultados obtidos e esperados).

Disponibilidade: 2.3.0

Exemplos

```
SELECT ST_AsText(ST_Normalize(ST_GeomFromText(
  'GEOMETRYCOLLECTION(
    POINT(2 3),
    MULTILINESTRING((0 0, 1 1),(2 2, 3 3)),
    POLYGON(
      (0 10,0 0,10 0,10 10,0 10),
      (4 2,2 2,2 4,4 4,4 2),
      (6 8,8 8,8 6,6 6,6 8)
    )
  )'
)));
                                                                                st_astext
-------------------------------------------------------------------------------------------------------------------------------

 GEOMETRYCOLLECTION(POLYGON((0 0,0 10,10 10,10 0,0 0),(6 6,8 6,8 8,6 8,6 6),(2 2,4 2,4 4,2  ↩
    4,2 2)),MULTILINESTRING((2 2,3 3),(0 0,1 1)),POINT(2 3))
(1 row)
```

Veja também

ST_Equals,

7.5.22 ST_Project

ST_Project — Returns a point projected from a start point by a distance and bearing (azimuth).

Synopsis

geometry **ST_Project**(geometry g1, float distance, float azimuth);
geometry **ST_Project**(geometry g1, geometry g2, float distance);
geography **ST_Project**(geography g1, float distance, float azimuth);
geography **ST_Project**(geography g1, geography g2, float distance);

Descrição

Returns a point projected from a point along a geodesic using a given distance and azimuth (bearing). This is known as the direct geodesic problem.

The two-point version uses the path from the first to the second point to implicitly define the azimuth and uses the distance as before.

The distance is given in meters. Negative values are supported.

The azimuth (also known as heading or bearing) is given in radians. It is measured clockwise from true north.

- North is azimuth zero (0 degrees)
- East is azimuth $\pi/2$ (90 degrees)
- South is azimuth π (180 degrees)
- West is azimuth $3\pi/2$ (270 degrees)

Negative azimuth values and values greater than 2π (360 degrees) are supported.

Disponibilidade: 2.0.0

Enhanced: 2.4.0 Allow negative distance and non-normalized azimuth.

Enhanced: 3.4.0 Allow geometry arguments and two-point form omitting azimuth.

Example: Projected point at 100,000 meters and bearing 45 degrees

```
SELECT ST_AsText(ST_Project('POINT(0 0)'::geography, 100000, radians(45.0)));
-----------------------------------------
 POINT(0.635231029125537 0.639472334729198)
```

Veja também

ST_Azimuth, ST_Distance, PostgreSQL function radians()

7.5.23 ST_QuantizeCoordinates

ST_QuantizeCoordinates — Sets least significant bits of coordinates to zero

Synopsis

geometry **ST_QuantizeCoordinates** (geometry g , int prec_x , int prec_y , int prec_z , int prec_m);

Descrição

`ST_QuantizeCoordinates` determines the number of bits (`N`) required to represent a coordinate value with a specified number of digits after the decimal point, and then sets all but the `N` most significant bits to zero. The resulting coordinate value will still round to the original value, but will have improved compressiblity. This can result in a significant disk usage reduction provided that the geometry column is using a compressible storage type. The function allows specification of a different number of digits after the decimal point in each dimension; unspecified dimensions are assumed to have the precision of the `x` dimension. Negative digits are interpreted to refer digits to the left of the decimal point, (i.e., `prec_x=-2` will preserve coordinate values to the nearest 100.

The coordinates produced by `ST_QuantizeCoordinates` are independent of the geometry that contains those coordinates and the relative position of those coordinates within the geometry. As a result, existing topological relationships between geometries are unaffected by use of this function. The function may produce invalid geometry when it is called with a number of digits lower than the intrinsic precision of the geometry.

Availability: 2.5.0

Technical Background

PostGIS stores all coordinate values as double-precision floating point integers, which can reliably represent 15 significant digits. However, PostGIS may be used to manage data that intrinsically has fewer than 15 significant digits. An example is TIGER data, which is provided as geographic coordinates with six digits of precision after the decimal point (thus requiring only nine significant digits of longitude and eight significant digits of latitude.)

When 15 significant digits are available, there are many possible representations of a number with 9 significant digits. A double precision floating point number uses 52 explicit bits to represent the significand (mantissa) of the coordinate. Only 30 bits are needed to represent a mantissa with 9 significant digits, leaving 22 insignificant bits; we can set their value to anything we like and still end up with a number that rounds to our input value. For example, the value 100.123456 can be represented by the floating point numbers closest to 100.123456000000, 100.123456000001, and 100.123456432199. All are equally valid, in that `ST_AsText(geom, 6)` will return the same result with any of these inputs. As we can set these bits to any value, `ST_QuantizeCoordinates` sets the 22 insignificant bits to zero. For a long coordinate sequence this creates a pattern of blocks of consecutive zeros that is compressed by PostgreSQL more effeciently.

Note
Only the on-disk size of the geometry is potentially affected by `ST_QuantizeCoordinates`. ST_MemSize, which reports the in-memory usage of the geometry, will return the the same value regardless of the disk space used by a geometry.

Exemplos

```
SELECT ST_AsText(ST_QuantizeCoordinates('POINT (100.123456 0)'::geometry, 4));
st_astext
-------------------------
POINT(100.123455047607 0)
```

```
WITH test AS (SELECT 'POINT (123.456789123456 123.456789123456)'::geometry AS geom)
SELECT
  digits,
  encode(ST_QuantizeCoordinates(geom, digits), 'hex'),
  ST_AsText(ST_QuantizeCoordinates(geom, digits))
FROM test, generate_series(15, -15, -1) AS digits;

digits  |                   encode                   |                st_astext
--------+--------------------------------------------+------------------------------------------ ↩

15      | 01010000005f9a72083cdd5e405f9a72083cdd5e40 | POINT(123.456789123456  ↩
    123.456789123456)
14      | 01010000005f9a72083cdd5e405f9a72083cdd5e40 | POINT(123.456789123456  ↩
    123.456789123456)
13      | 01010000005f9a72083cdd5e405f9a72083cdd5e40 | POINT(123.456789123456  ↩
    123.456789123456)
12      | 01010000005c9a72083cdd5e405c9a72083cdd5e40 | POINT(123.456789123456  ↩
    123.456789123456)
11      | 0101000000409a72083cdd5e40409a72083cdd5e40 | POINT(123.456789123456  ↩
    123.456789123456)
10      | 0101000000009a72083cdd5e40009a72083cdd5e40 | POINT(123.456789123455  ↩
    123.456789123455)
9       | 0101000000009072083cdd5e40009072083cdd5e40 | POINT(123.456789123418  ↩
    123.456789123418)
8       | 0101000000008072083cdd5e40008072083cdd5e40 | POINT(123.45678912336  ↩
    123.45678912336)
7       | 0101000000000070083cdd5e40000070083cdd5e40 | POINT(123.456789121032  ↩
    123.456789121032)
6       | 0101000000000040083cdd5e40000040083cdd5e40 | POINT(123.456789076328  ↩
    123.456789076328)
```

```
5       | 0101000000000000083cdd5e40000000083cdd5e40 | POINT(123.456789016724  ↩
    123.456789016724)
4       | 0101000000000000003cdd5e40000000003cdd5e40 | POINT(123.456787109375  ↩
    123.456787109375)
3       | 0101000000000000003cdd5e40000000003cdd5e40 | POINT(123.456787109375  ↩
    123.456787109375)
2       | 0101000000000000038dd5e400000000038dd5e40 | POINT(123.45654296875  ↩
    123.45654296875)
1       | 0101000000000000000dd5e400000000000dd5e40 | POINT(123.453125 123.453125)
0       | 0101000000000000000dc5e400000000000dc5e40 | POINT(123.4375 123.4375)
-1      | 0101000000000000000c05e400000000000c05e40 | POINT(123 123)
-2      | 01010000000000000000005e400000000000005e40 | POINT(120 120)
-3      | 010100000000000000000058400000000000005840 | POINT(96 96)
-4      | 010100000000000000000058400000000000005840 | POINT(96 96)
-5      | 010100000000000000000058400000000000005840 | POINT(96 96)
-6      | 010100000000000000000058400000000000005840 | POINT(96 96)
-7      | 010100000000000000000058400000000000005840 | POINT(96 96)
-8      | 010100000000000000000058400000000000005840 | POINT(96 96)
-9      | 010100000000000000000058400000000000005840 | POINT(96 96)
-10     | 010100000000000000000058400000000000005840 | POINT(96 96)
-11     | 010100000000000000000058400000000000005840 | POINT(96 96)
-12     | 010100000000000000000058400000000000005840 | POINT(96 96)
-13     | 010100000000000000000058400000000000005840 | POINT(96 96)
-14     | 010100000000000000000058400000000000005840 | POINT(96 96)
-15     | 010100000000000000000058400000000000005840 | POINT(96 96)
```

Veja também

ST_SnapToGrid

7.5.24 ST_RemovePoint

ST_RemovePoint — Remove a point from a linestring.

Synopsis

geometry **ST_RemovePoint**(geometry linestring, integer offset);

Descrição

Removes a point from a LineString, given its index (0-based). Useful for turning a closed line (ring) into an open linestring.

Enhanced: 3.2.0

Disponibilitade: 1.1.0

 This function supports 3d and will not drop the z-index.

Exemplos

Guarantees no lines are closed by removing the end point of closed lines (rings). Assumes geom is of type LINESTRING

```
UPDATE sometable
        SET geom = ST_RemovePoint(geom, ST_NPoints(geom) - 1)
        FROM sometable
        WHERE ST_IsClosed(geom);
```

Veja também

ST_AddPoint, ST_NPoints, ST_NumPoints

7.5.25 ST_RemoveRepeatedPoints

ST_RemoveRepeatedPoints — Returns a version of a geometry with duplicate points removed.

Synopsis

geometry **ST_RemoveRepeatedPoints**(geometry geom, float8 tolerance);

Descrição

Returns a version of the given geometry with duplicate consecutive points removed. The function processes only (Multi)LineStrings, (Multi)Polygons and MultiPoints but it can be called with any kind of geometry. Elements of GeometryCollections are processed individually. The endpoints of LineStrings are preserved.

If the *tolerance* parameter is provided, vertices within the tolerance distance of one another are considered to be duplicates.

Enhanced: 3.2.0

Disponibilidade: 2.2.0

 This function supports Polyhedral surfaces.

 This function supports 3d and will not drop the z-index.

Exemplos

```
SELECT ST_AsText( ST_RemoveRepeatedPoints( 'MULTIPOINT ((1 1), (2 2), (3 3), (2 2))'));
-------------------------
 MULTIPOINT(1 1,2 2,3 3)
```

```
SELECT ST_AsText( ST_RemoveRepeatedPoints( 'LINESTRING (0 0, 0 0, 1 1, 0 0, 1 1, 2 2)'));
---------------------------------
 LINESTRING(0 0,1 1,0 0,1 1,2 2)
```

Example: Collection elements are processed individually.

```
SELECT ST_AsText( ST_RemoveRepeatedPoints( 'GEOMETRYCOLLECTION (LINESTRING (1 1, 2 2, 2 2,  ↩
    3 3), POINT (4 4), POINT (4 4), POINT (5 5))'));
-----------------------------------------------------------------------------
 GEOMETRYCOLLECTION(LINESTRING(1 1,2 2,3 3),POINT(4 4),POINT(4 4),POINT(5 5))
```

Example: Repeated point removal with a distance tolerance.

```
SELECT ST_AsText( ST_RemoveRepeatedPoints( 'LINESTRING (0 0, 0 0, 1 1, 5 5, 1 1, 2 2)', 2)) ↩
    ;
-------------------------
 LINESTRING(0 0,5 5,2 2)
```

Veja também

ST_Simplify

7.5.26 ST_Reverse

ST_Reverse — Retorna a geometria com a ordem dos vértices revertida.

Synopsis

geometry **ST_Reverse**(geometry g1);

Descrição

Pode ser usado em qualquer geometria e reverte a ordem dos vértices.

Enhanced: 2.4.0 support for curves was introduced.

 This function supports 3d and will not drop the z-index.

 This function supports Polyhedral surfaces.

Exemplos

```
SELECT ST_AsText(geom) as line, ST_AsText(ST_Reverse(geom)) As reverseline
FROM
(SELECT ST_MakeLine(ST_Point(1,2),
                ST_Point(1,10)) As geom) as foo;
--result
                line         |     reverseline
---------------------+----------------------
LINESTRING(1 2,1 10) | LINESTRING(1 10,1 2)
```

7.5.27 ST_Segmentize

ST_Segmentize — Returns a modified geometry/geography having no segment longer than a given distance.

Synopsis

geometry **ST_Segmentize**(geometry geom, float max_segment_length);
geography **ST_Segmentize**(geography geog, float max_segment_length);

Descrição

Returns a modified geometry/geography having no segment longer than `max_segment_length`. Length is computed in 2D. Segments are always split into equal-length subsegments.

- For geometry, the maximum length is in the units of the spatial reference system.
- For geography, the maximum length is in meters. Distances are computed on the sphere. Added vertices are created along the spherical great-circle arcs defined by segment endpoints.

Note
This only shortens long segments. It does not lengthen segments shorter than the maximum length.

Warning
For inputs containing long segments, specifying a relatively short `max_segment_length` can cause a very large number of vertices to be added. This can happen unintentionally if the argument is specified accidentally as a number of segments, rather than a maximum length.

Disponibilidade: 1.2.2

Enhanced: 3.0.0 Segmentize geometry now produces equal-length subsegments

Enhanced: 2.3.0 Segmentize geography now produces equal-length subsegments

Melhorias: 2.1.0 suporte para geografia foi introduzido.

Changed: 2.1.0 As a result of the introduction of geography support, the usage `ST_Segmentize('LINESTRING(1 2, 3 4)', 0.5)` causes an ambiguous function error. The input needs to be properly typed as a geometry or geography. Use ST_GeomFromText, ST_GeogFromText or a cast to the required type (e.g. `ST_Segmentize('LINESTRING(1 2, 3 4)'::geometry, 0.5) )`

Exemplos

Segmentizing a line. Long segments are split evenly, and short segments are not split.

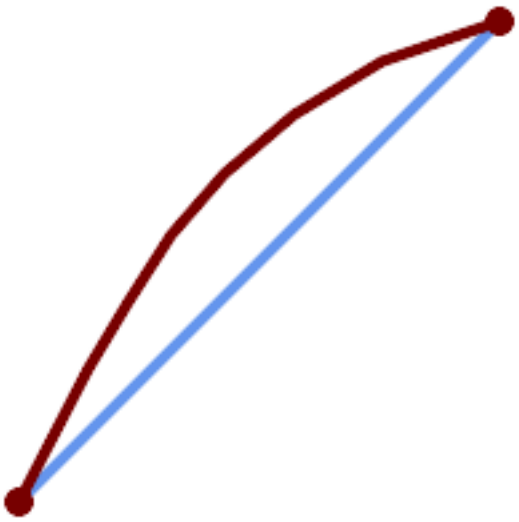
```
SELECT ST_AsText(ST_Segmentize(
    'MULTILINESTRING((0 0, 0 1, 0 9),(1 10, 1 18))'::geometry,
        5 ) );
--------------------------------------------------
MULTILINESTRING((0 0,0 1,0 5,0 9),(1 10,1 14,1 18))
```

Segmentizing a polygon:

```
SELECT ST_AsText(
        ST_Segmentize(('POLYGON((0 0, 0 8, 30 0, 0 0))'::geometry), 10));
----------------------------------------------------
POLYGON((0 0,0 8,7.5 6,15 4,22.5 2,30 0,20 0,10 0,0 0))
```

Segmentizing a geographic line, using a maximum segment length of 2000 kilometers. Vertices are added along the great-circle arc connecting the endpoints.

```
SELECT ST_AsText(
        ST_Segmentize(('LINESTRING (0 0, 60 60)'::geography), 2000000));
------------------------------------------------------------
LINESTRING(0 0,4.252632294621186 8.43596525986862,8.69579947419404  ↩
    16.824093489701564,13.550465473227048 25.107950473646188,19.1066053508691  ↩
    33.21091076089908,25.779290201459894 41.01711439406505,34.188839517966954  ↩
    48.337222885886,45.238153936612264 54.84733442373889,60 60)
```



A geographic line segmentized along a great circle arc

Veja também

ST_LineSubstring

7.5.28 ST_SetPoint

ST_SetPoint — Substitui ponto de uma linestring com um dado ponto.

Synopsis

geometry **ST_SetPoint**(geometry linestring, integer zerobasedposition, geometry point);

Descrição

Substitui ponto N de linstring com um dado ponto. Index é de base 0. Index negativo são contados atrasados, logo -1 é o último ponto. Isso é especialmente usado em causas tentando manter relações juntas quando um vértice se move.

Disponibilitade: 1.1.0

Atualizado 2.3.0: indexing negativo

This function supports 3d and will not drop the z-index.

Exemplos

```
--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.geom, ST_NumPoints(foo.geom) - 1, ST_GeomFromEWKT('POINT ↩
    (-1 1 3)')))
FROM (SELECT ST_GeomFromEWKT('LINESTRING(-1 2 3,-1 3 4, 5 6 7)') As geom) As foo;
           st_asewkt
```

```
-----------------------
LINESTRING(-1 2 3,-1 3 4,-1 1 3)

SELECT ST_AsText(ST_SetPoint(g, -3, p))
FROM ST_GEomFromText('LINESTRING(0 0, 1 1, 2 2, 3 3, 4 4)') AS g
        , ST_PointN(g,1) as p;
           st_astext
-----------------------
LINESTRING(0 0,1 1,0 0,3 3,4 4)
```

Veja também

ST_AddPoint, ST_NPoints, ST_NumPoints, ST_PointN, ST_RemovePoint

7.5.29 ST_ShiftLongitude

ST_ShiftLongitude — Shifts the longitude coordinates of a geometry between -180..180 and 0..360.

Synopsis

geometry **ST_ShiftLongitude**(geometry geom);

Descrição

Reads every point/vertex in a geometry, and shifts its longitude coordinate from -180..0 to 180..360 and vice versa if between these ranges. This function is symmetrical so the result is a 0..360 representation of a -180..180 data and a -180..180 representation of a 0..360 data.

Note
This is only useful for data with coordinates in longitude/latitude; e.g. SRID 4326 (WGS 84 geographic)

Warning
Pre-1.3.4 bug impediu de funcionar para MULTIPONTO. 1.3.4+ funciona com MULTIPONTO também.

 This function supports 3d and will not drop the z-index.

Melhorias: 2.0.0 suporte para superfícies poliédricas e TIN foi introduzido.

NOTA: esta função foi renomeada da "ST_Shift_Longitude" em 2.2.0

 This function supports Polyhedral surfaces.

 This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

Exemplos

```
--single point forward transformation
SELECT ST_AsText(ST_ShiftLongitude('SRID=4326;POINT(270 0)'::geometry))

st_astext
----------
POINT(-90 0)


--single point reverse transformation
SELECT ST_AsText(ST_ShiftLongitude('SRID=4326;POINT(-90 0)'::geometry))

st_astext
----------
POINT(270 0)


--for linestrings the functions affects only to the sufficient coordinates
SELECT ST_AsText(ST_ShiftLongitude('SRID=4326;LINESTRING(174 12, 182 13)'::geometry))

st_astext
----------
LINESTRING(174 12,-178 13)
```

Veja também

ST_WrapX

7.5.30 ST_WrapX

ST_WrapX — Envolve uma geometria em torno de um valor X.

Synopsis

geometry **ST_WrapX**(geometry geom, float8 wrap, float8 move);

Descrição

This function splits the input geometries and then moves every resulting component falling on the right (for negative 'move') or on the left (for positive 'move') of given 'wrap' line in the direction specified by the 'move' parameter, finally re-unioning the pieces together.

Note
Isto é útil para "recenter" entrada de long-lat para ter características de interesse não gerados de um lado para o outro.

Availability: 2.3.0 requires GEOS

 This function supports 3d and will not drop the z-index.

Exemplos

```
-- Move all components of the given geometries whose bounding box
-- falls completely on the left of x=0 to +360
select ST_WrapX(geom, 0, 360);

-- Move all components of the given geometries whose bounding box
-- falls completely on the left of x=-30 to +360
select ST_WrapX(geom, -30, 360);
```

Veja também

ST_ShiftLongitude

7.5.31 ST_SnapToGrid

ST_SnapToGrid — Rompe todos os pontos da geometria de entrada para uma rede regular.

Synopsis

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);

Descrição

Variante1,2,3: Rompe todos os pontos da geometria de entrada para a rede definida por sua origem e tamanho da célula. Remove pontos consecutivos caindo na mesma célula, finalmente retornando NULO se os pontos de saída não são suficientes para definir uma geometria do tipo dado. Geometrias colapsadas em uma coleção são desguarnecidas disso. Útil para reduzi a precisão.

Variante4: Introduzido 1.1.0 - Rompe todos os pontos da geometria de entrada para a rede definida por sua origem (o segundo argumento deve ser um ponto) e tamanhos de células. Especifica 0 como um tamanho para qualquer dimensão que você não quer romper para uma rede.

Note
A geometria de retorno pode perder sua simplicidade (veja ST_IsSimple).

Note
Antes de lançar 1.1.0, essa função sempre retornou uma geometria 2d. Começando em 1.1.0 a geometria de retorno terá a mesma dimensionalidade da entrada com maiores valores intocados de dimensão. Use a versão pegando um segundo argumento de geometria para definir todas as dimensões de rede.

Disponibilidade: 1.0.0RC1

Disponibilidade: 1.1.0 - suporte a Z e M

 This function supports 3d and will not drop the z-index.

Exemplos

```
--Snap your geometries to a precision grid of 10^-3
UPDATE mytable
   SET geom = ST_SnapToGrid(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
----------------------------------------------------------
 LINESTRING(-1.11 2.12 3 2.3456,4.11 3.24 3.1234 1.1111)
```

Veja também

ST_Snap, ST_AsEWKT, ST_AsText, ST_GeomFromText, ST_GeomFromEWKT, ST_Simplify

7.5.32 ST_Snap

ST_Snap — Rompe segmentos e vértices de geometria de entrada para vértices de uma geometria de referência.

Synopsis

geometry **ST_Snap**(geometry input, geometry reference, float tolerance);

Descrição

Snaps the vertices and segments of a geometry to another Geometry's vertices. A snap distance tolerance is used to control where snapping is performed. The result geometry is the input geometry with the vertices snapped. If no snapping occurs then the input geometry is returned unchanged.

Romper uma geometria para outra pode melhorar robusteza para operações de cobertura eliminando limites quase coincidentes (os quais causam problemas durante o sinal e cálculo de intersecção).

Romper muito pode resultar na criação de topologia inválida, então o número e localização dos vértices rompidos são decididos usando heurísticos para determinar quando é seguro romper. Entretanto, isso pode resultar em alguns rompimentos potencialmente omitidos.

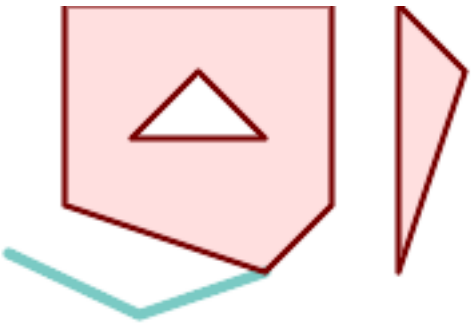
Note
A geometria devolvida pode perder sua simplicidade (veja ST_IsSimple) e validade (veja ST_IsValid).

Desempenhado pelo módulo GEOS.

Disponibilidade: 2.0.0

Exemplos

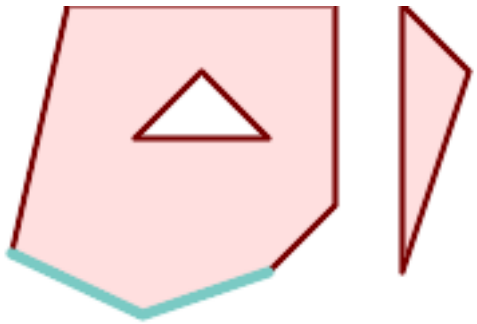
Um multi polígono apresentado com uma linestring (antes de qualquer rompimento)



Um multi polígono rompido para linestring para tolerância: 1.01 de distância. O novo multi polígono é mostrado com linestring de referência

```
SELECT ST_AsText(ST_Snap(poly,line,  ↩
    ST_Distance(poly,line)*1.01)) AS polysnapped
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;
```

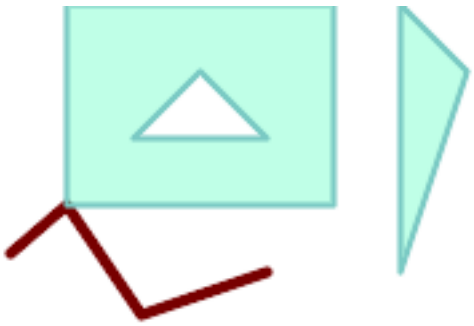
```
                            polysnapped
-----------------------------------------------------------------

 MULTIPOLYGON(((26 125,26 200,126 200,126 ↩
      125,101 100,26 125),
 (51 150,101 150,76 175,51 150)),((151  ↩
     100,151 200,176 175,151 100)))
```



Um multi polígono rompido para linestring para tolerância: 1.25 de distância. O novo multi polígono é mostrado com linestring de referência

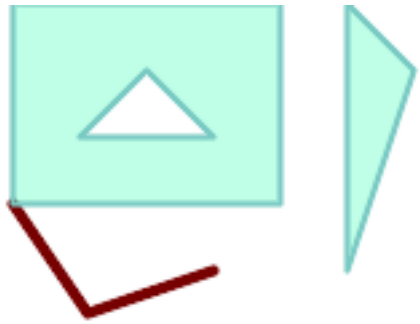
```
SELECT ST_AsText(
    ST_Snap(poly,line, ST_Distance(poly, ↩
     line)*1.25)
  ) AS polysnapped
FROM (SELECT
  ST_GeomFromText('MULTIPOLYGON(
    (( 26 125, 26 200, 126 200, 126 125,  ↩
     26 125 ),
      ( 51 150, 101 150, 76 175, 51 150 ) ↩
     ),
      (( 151 100, 151 200, 176 175, 151  ↩
     100 ))')) As poly,
       ST_GeomFromText('LINESTRING (5  ↩
     107, 54 84, 101 100)') As line
        ) As foo;
```

```
---------------------------------------- ↩    polysnapped
-----------------------------------------------------------------

MULTIPOLYGON(((5 107,26 200,126 200,126  ↩
     125,101 100,54 84,5 107),
(51 150,101 150,76 175,51 150)),((151  ↩
     100,151 200,176 175,151 100)))
```



A linestring rompida para o multi polígono original em tolerância de 1.01 de distância. As nova linestring é mostrada com multi polígono de referência

```
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)
```



A linestring rompida para o polígono original de tolerância 1.25 de distância. A nova linestring é mostrada com multi polígono de referência

```
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)
```

Veja também

ST_SnapToGrid

7.5.33 ST_SwapOrdinates

ST_SwapOrdinates — Retorna uma versão da geometria dada com os valores ordenados dados trocados.

Synopsis

geometry **ST_SwapOrdinates**(geometry geom, cstring ords);

Descrição

Retorna uma versão da geometria dada com as ordenadas dadas trocadas.

O parâmetro `ords` é uma string de 2-caracteres nomeando as ordenadas para trocar. Os nomes válidos são: x,y,z e m.

Disponibilidade: 2.2.0

 This method supports Circular Strings and Curves.

 This function supports 3d and will not drop the z-index.

 This function supports M coordinates.

 This function supports Polyhedral surfaces.

 This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

Exemplo

```
-- Scale M value by 2
SELECT ST_AsText(
  ST_SwapOrdinates(
    ST_Scale(
      ST_SwapOrdinates(g,'xm'),
      2, 1
    ),
  'xm')
) FROM ( SELECT 'POINT ZM (0 0 0 2)'::geometry g ) foo;
     st_astext
--------------------
 POINT ZM (0 0 0 4)
```

Veja também

ST_FlipCoordinates

7.6 Geometry Validation

7.6.1 ST_IsValid

ST_IsValid — Tests if a geometry is well-formed in 2D.

Synopsis

boolean **ST_IsValid**(geometry g);
boolean **ST_IsValid**(geometry g, integer flags);

Description

Tests if an ST_Geometry value is well-formed and valid in 2D according to the OGC rules. For geometries with 3 and 4 dimensions, the validity is still only tested in 2 dimensions. For geometries that are invalid, a PostgreSQL NOTICE is emitted providing details of why it is not valid.

For the version with the `flags` parameter, supported values are documented in ST_IsValidDetail This version does not print a NOTICE explaining invalidity.

For more information on the definition of geometry validity, refer to Section 4.4

Note
SQL-MM defines the result of ST_IsValid(NULL) to be 0, while PostGIS returns NULL.

Performed by the GEOS module.

The version accepting flags is available starting with 2.0.0.

This method implements the OGC Simple Features Implementation Specification for SQL 1.1.

This method implements the SQL/MM specification. SQL-MM 3: 5.1.9

Note
Neither OGC-SFS nor SQL-MM specifications include a flag argument for ST_IsValid. The flag is a PostGIS extension.

Examples

```
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
```

See Also

ST_IsSimple, ST_IsValidReason, ST_IsValidDetail,

7.6.2 ST_IsValidDetail

ST_IsValidDetail — Returns a `valid_detail` row stating if a geometry is valid or if not a reason and a location.

Synopsis

valid_detail **ST_IsValidDetail**(geometry geom, integer flags);

Description

Returns a `valid_detail` row, containing a boolean (`valid`) stating if a geometry is valid, a varchar (`reason`) stating a reason why it is invalid and a geometry (`location`) pointing out where it is invalid.

Useful to improve on the combination of ST_IsValid and ST_IsValidReason to generate a detailed report of invalid geometries.

The optional `flags` parameter is a bitfield. It can have the following values:

- 0: Use usual OGC SFS validity semantics.
- 1: Consider certain kinds of self-touching rings (inverted shells and exverted holes) as valid. This is also known as "the ESRI flag", since this is the validity model used by those tools. Note that this is invalid under the OGC model.

Performed by the GEOS module.

Availability: 2.0.0

Examples

```
--First 3 Rejects from a successful quintuplet experiment
SELECT gid, reason(ST_IsValidDetail(geom)), ST_AsText(location(ST_IsValidDetail(geom))) as ↩
    location
FROM
(SELECT ST_MakePolygon(ST_ExteriorRing(e.buff), array_agg(f.line)) As geom, gid
FROM (SELECT ST_Buffer(ST_Point(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_Point(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(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     |        |
```

See Also

ST_IsValid, ST_IsValidReason

7.6.3 ST_IsValidReason

ST_IsValidReason — Returns text stating if a geometry is valid, or a reason for invalidity.

Synopsis

text **ST_IsValidReason**(geometry geomA);
text **ST_IsValidReason**(geometry geomA, integer flags);

Description

Returns text stating if a geometry is valid, or if invalid a reason why.

Useful in combination with ST_IsValid to generate a detailed report of invalid geometries and reasons.

Allowed `flags` are documented in ST_IsValidDetail.

Performed by the GEOS module.

Availability: 1.4

Availability: 2.0 version taking flags.

Examples

```
-- invalid bow-tie polygon
SELECT ST_IsValidReason(
    'POLYGON ((100 200, 100 100, 200 200,
     200 100, 100 200))'::geometry) as validity_info;
validity_info
--------------------------
Self-intersection[150 150]
```

```
--First 3 Rejects from a successful quintuplet experiment
SELECT gid, ST_IsValidReason(geom) as validity_info
FROM
(SELECT ST_MakePolygon(ST_ExteriorRing(e.buff), array_agg(f.line)) As geom, gid
FROM (SELECT ST_Buffer(ST_Point(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_Point(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(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
```

See Also

ST_IsValid, ST_Summary

7.6.4 ST_MakeValid

ST_MakeValid — Attempts to make an invalid geometry valid without losing vertices.

Synopsis

geometry **ST_MakeValid**(geometry input);
geometry **ST_MakeValid**(geometry input, text params);

Description

The function attempts to create a valid representation of a given invalid geometry without losing any of the input vertices. Valid geometries are returned unchanged.

Supported inputs are: POINTS, MULTIPOINTS, LINESTRINGS, MULTILINESTRINGS, POLYGONS, MULTIPOLYGONS and GEOMETRYCOLLECTIONS containing any mix of them.

In case of full or partial dimensional collapses, the output geometry may be a collection of lower-to-equal dimension geometries, or a geometry of lower dimension.

Single polygons may become multi-geometries in case of self-intersections.

The `params` argument can be used to supply an options string to select the method to use for building valid geometry. The options string is in the format "method=linework|structure keepcollapsed=true|false". If no "params" argument is provided, the "linework" algorithm will be used as the default.

The "method" key has two values.

- "linework" is the original algorithm, and builds valid geometries by first extracting all lines, noding that linework together, then building a value output from the linework.
- "structure" is an algorithm that distinguishes between interior and exterior rings, building new geometry by unioning exterior rings, and then differencing all interior rings.

The "keepcollapsed" key is only valid for the "structure" algorithm, and takes a value of "true" or "false". When set to "false", geometry components that collapse to a lower dimensionality, for example a one-point linestring would be dropped.

Performed by the GEOS module.

Availability: 2.0.0

Enhanced: 2.0.1, speed improvements

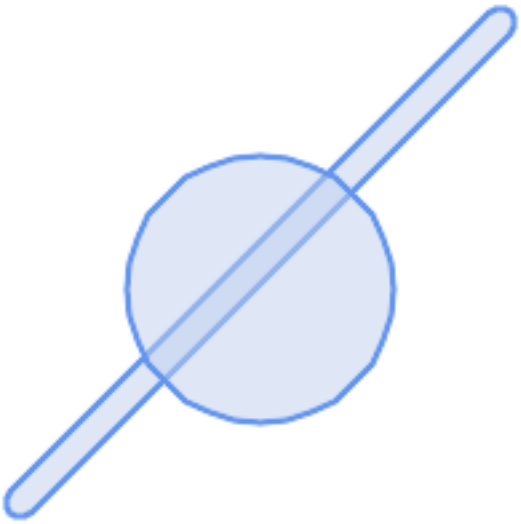
Enhanced: 2.1.0, added support for GEOMETRYCOLLECTION and MULTIPOINT.

Enhanced: 3.1.0, added removal of Coordinates with NaN values.

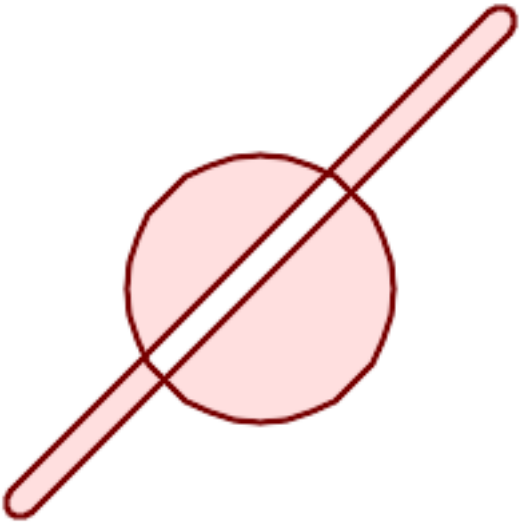
Enhanced: 3.2.0, added algorithm options, 'linework' and 'structure' which requires GEOS >= 3.10.0.

 This function supports 3d and will not drop the z-index.

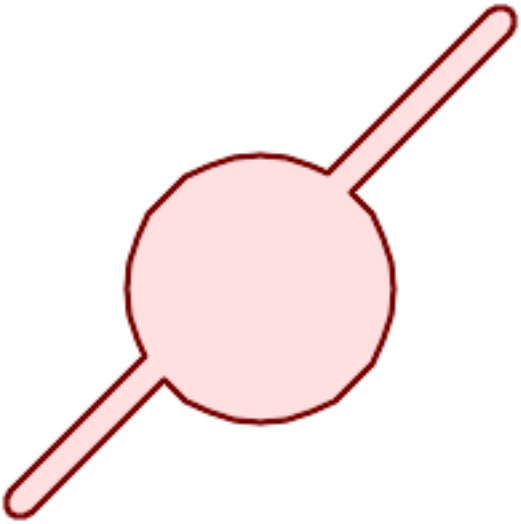
Examples



before_geom: MULTIPOLYGON of 2 overlapping polygons

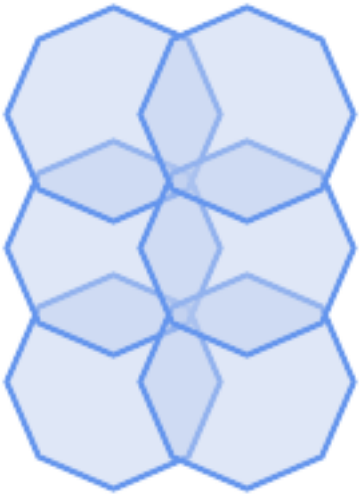


after_geom: MULTIPOLYGON of 4 non-overlapping polygons

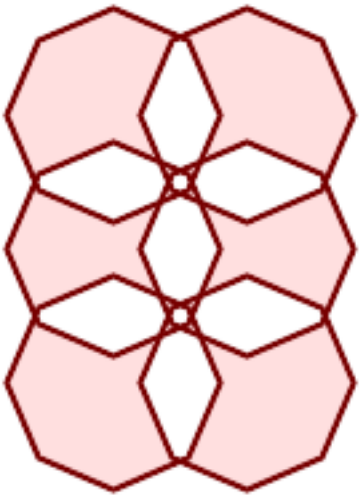


after_geom_structure: MULTIPOLYGON of 1 non-overlapping polygon

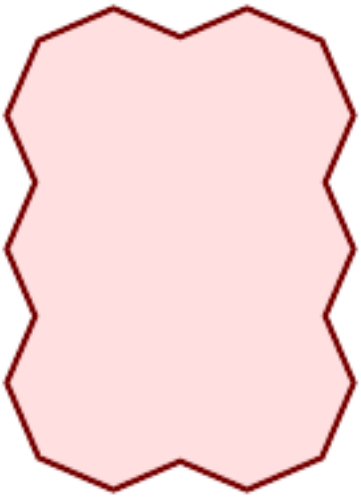
```
SELECT f.geom AS before_geom, ST_MakeValid(f.geom) AS after_geom, ST_MakeValid(f.geom,  ↩
    'method=structure') AS after_geom_structure
FROM (SELECT 'MULTIPOLYGON(((186 194,187 194,188 195,189 195,190 195,
191 195,192 195,193 194,194 194,194 194,193 195,192 195,191
```



before_geom: MULTIPOLYGON of 6 overlapping polygons



after_geom: MULTIPOLYGON of 14 Non-overlapping polygons



after_geom_structure: MULTIPOLYGON of 1 Non-overlapping polygon

```
SELECT c.geom AS before_geom,
                    ST_MakeValid(c.geom) AS after_geom,
                    ST_MakeValid(c.geom, 'method=structure') AS after_geom_structure
        FROM (SELECT 'MULTIPOLYGON(((91 50,79 22,51 10,23 22,11 50,23 78,51 90,79 78,91 ↩
```

Examples

```
SELECT ST_AsText(ST_MakeValid(
    'LINESTRING(0 0, 0 0)',
    'method=structure keepcollapsed=true'
    ));

 st_astext
------------
 POINT(0 0)


SELECT ST_AsText(ST_MakeValid(
    'LINESTRING(0 0, 0 0)',
    'method=structure keepcollapsed=false'
    ));

    st_astext
------------------
 LINESTRING EMPTY
```

See Also

ST_IsValid, ST_GeomCollFromText, ST_CollectionExtract

7.7 Spatial Reference System Functions

7.7.1 ST_InverseTransformPipeline

ST_InverseTransformPipeline — Return a new geometry with coordinates transformed to a different spatial reference system using the inverse of a defined coordinate transformation pipeline.

Synopsis

geometry **ST_InverseTransformPipeline**(geometry geom, text pipeline, integer to_srid);

Description

Return a new geometry with coordinates transformed to a different spatial reference system using a defined coordinate transformation pipeline to go in the inverse direction.

Refer to ST_TransformPipeline for details on writing a transformation pipeline.

Availability: 3.4.0

The SRID of the input geometry is ignored, and the SRID of the output geometry will be set to zero unless a value is provided via the optional `to_srid` parameter. When using ST_TransformPipeline the pipeline is executed in a forward direction. Using `ST_InverseTransformPipeline()` the pipeline is executed in the inverse direction.

Transforms using pipelines are a specialised version of ST_Transform. In most cases `ST_Transform` will choose the correct operations to convert between coordinate systems, and should be preferred.

Examples

Change WGS 84 long lat to UTM 31N using the EPSG:16031 conversion

```
-- Inverse direction
SELECT ST_AsText(ST_InverseTransformPipeline('POINT(426857.9877165967 5427937.523342293)':: ↩
    geometry,
  'urn:ogc:def:coordinateOperation:EPSG::16031')) AS wgs_geom;

          wgs_geom
----------------------------
 POINT(2 48.99999999999999)
(1 row)
```

GDA2020 example.

```
-- using ST_Transform with automatic selection of a conversion pipeline.
SELECT ST_AsText(ST_Transform('SRID=4939;POINT(143.0 -37.0)'::geometry, 7844)) AS  ↩
    gda2020_auto;

                 gda2020_auto
---------------------------------------------
 POINT(143.00000635638918 -36.999986706128176)
(1 row)
```

See Also

ST_Transform, ST_TransformPipeline

7.7.2 ST_SetSRID

ST_SetSRID — Set the SRID on a geometry.

Synopsis

geometry **ST_SetSRID**(geometry geom, integer srid);

Description

Sets the SRID on a geometry to a particular integer value. Useful in constructing bounding boxes for queries.

Note
This function does not transform the geometry coordinates in any way - it simply sets the meta data defining the spatial reference system the geometry is assumed to be in. Use ST_Transform if you want to transform the geometry into a new projection.


This method implements the OGC Simple Features Implementation Specification for SQL 1.1.


This method supports Circular Strings and Curves.

Examples

-- 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)
```

See Also

Section 4.5, ST_SRID, ST_Transform, UpdateGeometrySRID

7.7.3 ST_SRID

ST_SRID — Returns the spatial reference identifier for a geometry.

Synopsis

integer **ST_SRID**(geometry g1);

Description

Returns the spatial reference identifier for the ST_Geometry as defined in spatial_ref_sys table. Section 4.5

Note
spatial_ref_sys table is a table that catalogs all spatial reference systems known to PostGIS and is used for transformations from one spatial reference system to another. So verifying you have the right spatial reference system identifier is important if you plan to ever transform your geometries.


This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.1


This method implements the SQL/MM specification. SQL-MM 3: 5.1.5


This method supports Circular Strings and Curves.

Examples

```
SELECT ST_SRID(ST_GeomFromText('POINT(-71.1043 42.315)',4326));
    --result
    4326
```

See Also

Section 4.5, ST_SetSRID, ST_Transform, ST_SRID, ST_SRID

7.7.4 ST_Transform

ST_Transform — Return a new geometry with coordinates transformed to a different spatial reference system.

Synopsis

geometry **ST_Transform**(geometry g1, integer srid);
geometry **ST_Transform**(geometry geom, text to_proj);
geometry **ST_Transform**(geometry geom, text from_proj, text to_proj);
geometry **ST_Transform**(geometry geom, text from_proj, integer to_srid);

Description

Returns a new geometry with its coordinates transformed to a different spatial reference system. The destination spatial reference `to_srid` may be identified by a valid SRID integer parameter (i.e. it must exist in the `spatial_ref_sys` table). Alternatively, a spatial reference defined as a PROJ.4 string can be used for `to_proj` and/or `from_proj`, however these methods are not optimized. If the destination spatial reference system is expressed with a PROJ.4 string instead of an SRID, the SRID of the output geometry will be set to zero. With the exception of functions with `from_proj`, input geometries must have a defined SRID.

ST_Transform is often confused with ST_SetSRID. ST_Transform actually changes the coordinates of a geometry from one spatial reference system to another, while ST_SetSRID() simply changes the SRID identifier of the geometry.

ST_Transform automatically selects a suitable conversion pipeline given the source and target spatial reference systems. To use a specific conversion method, use ST_TransformPipeline.

Note
Requires PostGIS be compiled with PROJ support. Use PostGIS_Full_Version to confirm you have PROJ support compiled in.

Note
If using more than one transformation, it is useful to have a functional index on the commonly used transformations to take advantage of index usage.

Note
Prior to 1.3.4, this function crashes if used with geometries that contain CURVES. This is fixed in 1.3.4+

Enhanced: 2.0.0 support for Polyhedral surfaces was introduced.

Enhanced: 2.3.0 support for direct PROJ.4 text was introduced.


This method implements the SQL/MM specification. SQL-MM 3: 5.1.6


This method supports Circular Strings and Curves.


This function supports Polyhedral surfaces.

Examples

Change Massachusetts 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)
```

Example of creating a partial functional index. For tables where you are not sure all the geometries will be filled in, its best to use a partial index that leaves out null geometries which will both conserve space and make your index smaller and more efficient.

```
CREATE INDEX idx_geom_26986_parcels
  ON parcels
  USING gist
  (ST_Transform(geom, 26986))
  WHERE geom IS NOT NULL;
```

Examples of using PROJ.4 text to transform with custom spatial references.

```
-- Find intersection of two polygons near the North pole, using a custom Gnomic projection
-- See http://boundlessgeo.com/2012/02/flattening-the-peel/
 WITH data AS (
   SELECT
     ST_GeomFromText('POLYGON((170 50,170 72,-130 72,-130 50,170 50))', 4326) AS p1,
     ST_GeomFromText('POLYGON((-170 68,-170 90,-141 90,-141 68,-170 68))', 4326) AS p2,
     '+proj=gnom +ellps=WGS84 +lat_0=70 +lon_0=-160 +no_defs'::text AS gnom
 )
 SELECT ST_AsText(
   ST_Transform(
     ST_Intersection(ST_Transform(p1, gnom), ST_Transform(p2, gnom)),
   gnom, 4326))
 FROM data;
                                          st_astext
 ------------------------------------------------------------------------------
  POLYGON((-170 74.053793645338,-141 73.4268621378904,-141 68,-170 68,-170 74.053793645338) ↩
      )
```

Configuring transformation behavior

Sometimes coordinate transformation involving a grid-shift can fail, for example if PROJ.4 has not been built with grid-shift files or the coordinate does not lie within the range for which the grid shift is defined. By default, PostGIS will throw an error if a

grid shift file is not present, but this behavior can be configured on a per-SRID basis either by testing different `to_proj` values of PROJ.4 text, or altering the `proj4text` value within the `spatial_ref_sys` table.

For example, the proj4text parameter +datum=NAD87 is a shorthand form for the following +nadgrids parameter:

```
+nadgrids=@conus,@alaska,@ntv2_0.gsb,@ntv1_can.dat
```

The @ prefix means no error is reported if the files are not present, but if the end of the list is reached with no file having been appropriate (ie. found and overlapping) then an error is issued.

If, conversely, you wanted to ensure that at least the standard files were present, but that if all files were scanned without a hit a null transformation is applied you could use:

```
+nadgrids=@conus,@alaska,@ntv2_0.gsb,@ntv1_can.dat,null
```

The null grid shift file is a valid grid shift file covering the whole world and applying no shift. So for a complete example, if you wanted to alter PostGIS so that transformations to SRID 4267 that didn't lie within the correct range did not throw an ERROR, you would use the following:

```
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;
```

See Also

Section 4.5, ST_SetSRID, ST_SRID, UpdateGeometrySRID, ST_TransformPipeline

7.7.5 ST_TransformPipeline

ST_TransformPipeline — Return a new geometry with coordinates transformed to a different spatial reference system using a defined coordinate transformation pipeline.

Synopsis

geometry **ST_TransformPipeline**(geometry g1, text pipeline, integer to_srid);

Description

Return a new geometry with coordinates transformed to a different spatial reference system using a defined coordinate transformation pipeline.

Transformation pipelines are defined using any of the following string formats:

- `urn:ogc:def:coordinateOperation:AUTHORITY::CODE`. Note that a simple `EPSG:CODE` string does not uniquely identify a coordinate operation: the same EPSG code can be used for a CRS definition.
- A PROJ pipeline string of the form: `+proj=pipeline ...`. Automatic axis normalisation will not be applied, and if necessary the caller will need to add an additional pipeline step, or remove `axisswap` steps.
- Concatenated operations of the form: `urn:ogc:def:coordinateOperation,coordinateOperation:EPSG::3895,c`

Availability: 3.4.0

The SRID of the input geometry is ignored, and the SRID of the output geometry will be set to zero unless a value is provided via the optional `to_srid` parameter. When using `ST_TransformPipeline()` the pipeline is executed in a forward direction. Using ST_InverseTransformPipeline the pipeline is executed in the inverse direction.

Transforms using pipelines are a specialised version of ST_Transform. In most cases `ST_Transform` will choose the correct operations to convert between coordinate systems, and should be preferred.

Examples

Change WGS 84 long lat to UTM 31N using the EPSG:16031 conversion

```
-- Forward direction
SELECT ST_AsText(ST_TransformPipeline('SRID=4326;POINT(2 49)'::geometry,
  'urn:ogc:def:coordinateOperation:EPSG::16031') AS utm_geom);

                 utm_geom
-------------------------------------------
 POINT(426857.9877165967 5427937.523342293)
(1 row)

-- Inverse direction
SELECT ST_AsText(ST_InverseTransformPipeline('POINT(426857.9877165967 5427937.523342293)':: ↩
    geometry,
  'urn:ogc:def:coordinateOperation:EPSG::16031')) AS wgs_geom;

          wgs_geom
----------------------------
 POINT(2 48.99999999999999)
(1 row)
```

GDA2020 example.

```
-- using ST_Transform with automatic selection of a conversion pipeline.
SELECT ST_AsText(ST_Transform('SRID=4939;POINT(143.0 -37.0)'::geometry, 7844)) AS  ↩
    gda2020_auto;

                gda2020_auto
---------------------------------------------
 POINT(143.00000635638918 -36.999986706128176)
(1 row)

-- using a defined conversion (EPSG:8447)
SELECT ST_AsText(ST_TransformPipeline('SRID=4939;POINT(143.0 -37.0)'::geometry,
  'urn:ogc:def:coordinateOperation:EPSG::8447')) AS gda2020_code;

                 gda2020_code
---------------------------------------------
 POINT(143.0000063280214 -36.999986718287545)
(1 row)

-- using a PROJ pipeline definition matching EPSG:8447, as returned from
-- 'projinfo -s EPSG:4939 -t EPSG:7844'.
-- NOTE: any 'axisswap' steps must be removed.
SELECT ST_AsText(ST_TransformPipeline('SRID=4939;POINT(143.0 -37.0)'::geometry,
  '+proj=pipeline
   +step +proj=unitconvert +xy_in=deg +xy_out=rad
   +step +proj=hgridshift +grids=au_icsm_GDA94_GDA2020_conformal_and_distortion.tif
   +step +proj=unitconvert +xy_in=rad +xy_out=deg')) AS gda2020_pipeline;

                 gda2020_pipeline
---------------------------------------------
 POINT(143.0000063280214 -36.999986718287545)
(1 row)
```

See Also

ST_Transform, ST_InverseTransformPipeline

7.7.6 postgis_srs_codes

postgis_srs_codes — Return the list of SRS codes associated with the given authority.

Synopsis

setof text **postgis_srs_codes**(text auth_name);

Description

Returns a set of all `auth_srid` for the given `auth_name`.

Availability: 3.4.0

Proj version 6+

Examples

List the first ten codes associated with the EPSG authority.

```
SELECT * FROM postgis_srs_codes('EPSG') LIMIT 10;

 postgis_srs_codes
-------------------
 2000
 20004
 20005
 20006
 20007
 20008
 20009
 2001
 20010
 20011
```

See Also

postgis_srs, postgis_srs_all, postgis_srs_search

7.7.7 postgis_srs

postgis_srs — Return a metadata record for the requested authority and srid.

Synopsis

setof record **postgis_srs**(text auth_name, text auth_srid);

Description

Returns a metadata record for the requested `auth_srid` for the given `auth_name`. The record will have the `auth_name`, `auth_srid`, `srname`, `srtext`, `proj4text`, and the corners of the area of usage, `point_sw` and `point_ne`.

Availability: 3.4.0

Proj version 6+

Examples

Get the metadata for EPSG:3005.

```
SELECT * FROM postgis_srs('EPSG', '3005');

auth_name | EPSG
auth_srid | 3005
srname    | NAD83 / BC Albers
srtext    | PROJCS["NAD83 / BC Albers", ... ]]
proj4text | +proj=aea +lat_0=45 +lon_0=-126 +lat_1=50 +lat_2=58.5 +x_0=1000000 +y_0=0 + ↩
    datum=NAD83 +units=m +no_defs +type=crs
point_sw  | 0101000020E6100000E17A14AE476161C00000000000204840
point_ne  | 0101000020E610000085EB51B81E855CC0E17A14AE47014E40
```

See Also

postgis_srs_codes, postgis_srs_all, postgis_srs_search

7.7.8 postgis_srs_all

postgis_srs_all — Return metadata records for every spatial reference system in the underlying Proj database.

Synopsis

setof record **postgis_srs_all**(void);

Description

Returns a set of all metadata records in the underlying Proj database. The records will have the `auth_name`, `auth_srid`, `srname`, `srtext`, `proj4text`, and the corners of the area of usage, `point_sw` and `point_ne`.

Availability: 3.4.0

Proj version 6+

Examples

Get the first 10 metadata records from the Proj database.

```
SELECT auth_name, auth_srid, srname FROM postgis_srs_all() LIMIT 10;

 auth_name | auth_srid |                  srname
-----------+-----------+-----------------------------------------
 EPSG      | 2000      | Anguilla 1957 / British West Indies Grid
 EPSG      | 20004     | Pulkovo 1995 / Gauss-Kruger zone 4
 EPSG      | 20005     | Pulkovo 1995 / Gauss-Kruger zone 5
 EPSG      | 20006     | Pulkovo 1995 / Gauss-Kruger zone 6
 EPSG      | 20007     | Pulkovo 1995 / Gauss-Kruger zone 7
 EPSG      | 20008     | Pulkovo 1995 / Gauss-Kruger zone 8
 EPSG      | 20009     | Pulkovo 1995 / Gauss-Kruger zone 9
 EPSG      | 2001      | Antigua 1943 / British West Indies Grid
 EPSG      | 20010     | Pulkovo 1995 / Gauss-Kruger zone 10
 EPSG      | 20011     | Pulkovo 1995 / Gauss-Kruger zone 11
```

See Also

postgis_srs_codes, postgis_srs, postgis_srs_search

7.7.9 postgis_srs_search

postgis_srs_search — Return metadata records for projected coordinate systems that have areas of useage that fully contain the bounds parameter.

Synopsis

setof record **postgis_srs_search**(geometry bounds, text auth_name=EPSG);

Description

Return a set of metadata records for projected coordinate systems that have areas of useage that fully contain the bounds parameter. Each record will have the `auth_name`, `auth_srid`, `srname`, `srtext`, `proj4text`, and the corners of the area of usage, `point_sw` and `point_ne`.

The search only looks for projected coordinate systems, and is intended for users to explore the possible systems that work for the extent of their data.

Availability: 3.4.0

Proj version 6+

Examples

Search for projected coordinate systems in Louisiana.

```
SELECT auth_name, auth_srid, srname,
  ST_AsText(point_sw) AS point_sw,
  ST_AsText(point_ne) AS point_ne
FROM postgis_srs_search('SRID=4326;LINESTRING(-90 30, -91 31)')
LIMIT 3;

 auth_name | auth_srid |                srname                |      point_sw       |       ↩
    point_ne
-----------+-----------+--------------------------------------+---------------------+---------------

 EPSG      | 2801      | NAD83(HARN) / Louisiana South        | POINT(-93.94 28.85) | POINT ↩
    (-88.75 31.07)
 EPSG      | 3452      | NAD83 / Louisiana South (ftUS)       | POINT(-93.94 28.85) | POINT ↩
    (-88.75 31.07)
 EPSG      | 3457      | NAD83(HARN) / Louisiana South (ftUS) | POINT(-93.94 28.85) | POINT ↩
    (-88.75 31.07)
```

Scan a table for max extent and find projected coordinate systems that might suit.

```
WITH ext AS (
  SELECT ST_Extent(geom) AS geom, Max(ST_SRID(geom)) AS srid
  FROM foo
)
SELECT auth_name, auth_srid, srname,
  ST_AsText(point_sw) AS point_sw,
  ST_AsText(point_ne) AS point_ne
FROM ext
CROSS JOIN postgis_srs_search(ST_SetSRID(ext.geom, ext.srid))
LIMIT 3;
```

See Also

postgis_srs_codes, postgis_srs_all, postgis_srs

7.8 Geometry Input

7.8.1 Well-Known Text (WKT)

7.8.1.1 ST_BdPolyFromText

ST_BdPolyFromText — Constrói um polígono dada uma coleção arbitrária de linestrings fechadas como uma representação de texto de uma multilinestring bem conhecida.

Synopsis

geometria **ST_BdPolyFromText**(texto WKT, inteiro srid);

Descrição

Constrói um polígono dada uma coleção arbitrária de linestrings fechadas como uma representação de texto de uma multilinestring bem conhecida.

Note
Lança um erro se WKT não é uma MULTILINESTRING. Lança um erro se a saída não é um MULTIPOLÍGONO; use ST_BdMPolyFromText nesse caso, ou veja ST_BuildArea() para uma aproximação postgis-specific.

This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2

Desempenhado pelo módulo GEOS.

Disponibilidade: 1.1.0

Veja também.

ST_BuildArea, ST_BdMPolyFromText

7.8.1.2 ST_BdMPolyFromText

ST_BdMPolyFromText — Constrói um polígono dada uma coleção arbitrária de linestrings fechadas como uma representação de texto de uma multilinestring bem conhecida.

Synopsis

geometria **ST_BdMPolyFromText**(text WKT, inteiro srid);

Descrição

Constrói um um polígono dada uma coleção arbitrária de linestrings, polígonos, multilinestrings fechados como uma representação de texto bem conhecida.

Note
Lança um erro se WKT não é uma MULTILINESTRING. Força a saída MULTIPOLÍGONO mesmo quando o resultado não é composto somente por um POLÍGONO único; use ST_BdPolyFromText se você tem certeza de que um único POLÍGONO irá resultar de uma operação, ou veja ST_BuildArea() para uma aproximação postgis-specific.

 This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2

Desempenhado pelo módulo GEOS.

Disponibilidade: 1.1.0

Veja também.

ST_BuildArea, ST_BdPolyFromText

7.8.1.3 ST_GeogFromText

ST_GeogFromText — Retorna um valor de geografia específico de uma representação bem conhecida de texto ou estendida (WKT).

Synopsis

geography **ST_GeogFromText**(texto EWKT);

Descrição

Retorna um objeto de geografia de um texto bem conhecido ou representação estendida bem conhecida. SRID 4326 é suposta se não for especificada. Isso é um heterônimo para ST_GeographyFromText. Os pontos são sempre expressados em uma forma long lat.

Exemplos

```
--- converting lon lat coords to geography
ALTER TABLE sometable ADD COLUMN geog geography(POINT,4326);
UPDATE sometable SET geog = ST_GeogFromText('SRID=4326;POINT(' || lon || ' ' || lat || ')') ↩
    ;

--- specify a geography point using EPSG:4267, NAD27
SELECT ST_AsEWKT(ST_GeogFromText('SRID=4267;POINT(-77.0092 38.889588)'));
```

Veja também.

ST_AsText, ST_GeographyFromText

7.8.1.4 ST_GeographyFromText

ST_GeographyFromText — Retorna um valor de geografia específico de uma representação bem conhecida de texto ou estendida (WKT).

Synopsis

geography **ST_GeographyFromText**(texto EWKT);

Descrição

Retorna um objeto de geografia de um texto bem conhecido ou representação estendida bem conhecida. SRID 4326 é suposta se não for especificada

Veja também.

ST_GeogFromText, ST_AsText

7.8.1.5 ST_GeomCollFromText

ST_GeomCollFromText — Makes a collection Geometry from collection WKT with the given SRID. If SRID is not given, it defaults to 0.

Synopsis

geometry **ST_GeomCollFromText**(text WKT, integer srid);
geometry **ST_GeomCollFromText**(text WKT);

Descrição

Makes a collection Geometry from the Well-Known-Text (WKT) representation with the given SRID. If SRID is not given, it defaults to 0.

OGC SPEC 3.2.6.2 - opção SRID é de uma suíte de conformação

Retorna nula se a WKT não for uma GEOMETRYCOLLECTION

Note
se você não tem total certeza de que todas suas geometrias WKT são coleções, não use essa função. Ela é mais devagar que a ST_GeomFromText, já que adiciona um passo de validação adicional.


This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2


This method implements the SQL/MM specification.

Exemplos

```
SELECT ST_GeomCollFromText('GEOMETRYCOLLECTION(POINT(1 2),LINESTRING(1 2, 3 4))');
```

Veja também.

ST_GeomFromText, ST_SRID

7.8.1.6 ST_GeomFromEWKT

ST_GeomFromEWKT — Retorna um valor ST_Geometry especifico da representação de texto estendida bem conhecida (EWKT).

Synopsis

geometria **ST_GeomFromEWKT**(texto EWKT);

Descrição

Constrói um objeto PostGIS ST_Geometry da representação de texto estendida bem conhecida OGC (EWKT).

Note
O formato EWKT não é um padrão OGC, mas um formato específico PostGIS que inclui o identificador de sistema de referência espacial (SRID).

Melhorias: 2.0.0 suporte para superfícies poliédricas e TIN foi introduzido.

 This function supports 3d and will not drop the z-index.

 This method supports Circular Strings and Curves.

 This function supports Polyhedral surfaces.

 This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

Exemplos

```
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,
-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))
)');
```

Veja também.

ST_AsEWKT, ST_GeomFromText

7.8.1.7 ST_GeomFromMARC21

ST_GeomFromMARC21 — Takes MARC21/XML geographic data as input and returns a PostGIS geometry object.

Synopsis

geometry **ST_GeomFromMARC21** (text marcxml);

Descrição

This function creates a PostGIS geometry from a MARC21/XML record, which can contain a `POINT` or a `POLYGON`. In case of multiple geographic data entries in the same MARC21/XML record, a `MULTIPOINT` or `MULTIPOLYGON` will be returned. If the record contains mixed geometry types, a `GEOMETRYCOLLECTION` will be returned. It returns NULL if the MARC21/XML record does not contain any geographic data (datafield:034).

LOC MARC21/XML versions supported:

- MARC21/XML 1.1

Availability: 3.3.0, requires libxml2 2.6+

Note
The MARC21/XML Coded Cartographic Mathematical Data currently does not provide any means to describe the Spatial Reference System of the encoded coordinates, so this function will always return a geometry with `SRID 0`.

Note
Returned `POLYGON` geometries will always be clockwise oriented.

Exemplos

Converting MARC21/XML geographic data containing a single `POINT` encoded as `hddd.dddddd`

```
SELECT
                ST_AsText(
                    ST_GeomFromMARC21('
                        <record xmlns="http://www.loc.gov/MARC21/slim">
                            <leader
>00000nz a2200000nc 4500</leader>
                            <controlfield tag="001"
>040277569</controlfield>
                            <datafield tag="034" ind1=" " ind2=" ">
                                <subfield code="d"
>W004.500000</subfield>
                                <subfield code="e"
>W004.500000</subfield>
                                <subfield code="f"
>N054.250000</subfield>
                                <subfield code="g"
>N054.250000</subfield>
                            </datafield>
                        </record
>'));

                st_astext
                -------------------
                POINT(-4.5 54.25)
                (1 row)
```

Converting MARC21/XML geographic data containing a single `POLYGON` encoded as `hdddmmss`

```
SELECT
                ST_AsText(
                    ST_GeomFromMARC21('
                        <record xmlns="http://www.loc.gov/MARC21/slim">
                            <leader
>01062cem a2200241 a 4500</leader>
                            <controlfield tag="001"
>   84696781 </controlfield>
                            <datafield tag="034" ind1="1" ind2=" ">
                                <subfield code="a"
>a</subfield>
                                <subfield code="b"
>50000</subfield>
                                <subfield code="d"
>E0130600</subfield>
                                <subfield code="e"
>E0133100</subfield>
                                <subfield code="f"
>N0523900</subfield>
                                <subfield code="g"
>N0522300</subfield>
                            </datafield>
                        </record
>'));

                st_astext
                ----------------------------------------------------------------------------------

                POLYGON((13.1 52.65,13.516666666666667 52.65,13.516666666666667  ↩
                    52.38333333333333,13.1 52.38333333333333,13.1 52.65))
```

```
                (1 row)
```

Converting MARC21/XML geographic data containing a `POLYGON` and a `POINT`:

```
SELECT
                ST_AsText(
                    ST_GeomFromMARC21('
                <record xmlns="http://www.loc.gov/MARC21/slim">
                    <datafield tag="034" ind1="1" ind2=" ">
                        <subfield code="a"
>a</subfield>
                        <subfield code="b"
>50000</subfield>
                        <subfield code="d"
>E0130600</subfield>
                        <subfield code="e"
>E0133100</subfield>
                        <subfield code="f"
>N0523900</subfield>
                        <subfield code="g"
>N0522300</subfield>
                    </datafield>
                    <datafield tag="034" ind1=" " ind2=" ">
                        <subfield code="d"
>W004.500000</subfield>
                        <subfield code="e"
>W004.500000</subfield>
                        <subfield code="f"
>N054.250000</subfield>
                        <subfield code="g"
>N054.250000</subfield>
                    </datafield>
                </record
>'));
                                                                                st_astext ↩

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

                GEOMETRYCOLLECTION(POLYGON((13.1 52.65,13.516666666666667  ↩
                    52.65,13.516666666666667 52.38333333333333,13.1 52.38333333333333,13.1  ↩
                    52.65)),POINT(-4.5 54.25))
                (1 row)
```

Veja também.

ST_AsMARC21

7.8.1.8 ST_GeometryFromText

ST_GeometryFromText — Retorna um valor ST_Geometry especifico da representação de texto estendida bem conhecida (EWKT). Isso é um heterônimo para ST_GeomFromText

Synopsis

geometry **ST_GeometryFromText**(text WKT);
geometry **ST_GeometryFromText**(text WKT, integer srid);

Descrição

This method implements the OGC Simple Features Implementation Specification for SQL 1.1.

This method implements the SQL/MM specification. SQL-MM 3: 5.1.40

Veja também.

ST_GeomFromText

7.8.1.9 ST_GeomFromText

ST_GeomFromText — Retorna um valor ST_Geometry específico da representação de texto bem conhecida (WKT).

Synopsis

geometry **ST_GeomFromText**(text WKT);
geometry **ST_GeomFromText**(text WKT, integer srid);

Descrição

Constrói um objeto PostGIS ST_Geometry de uma representação de texto bem conhecida OGC.

Note
Existem duas variantes da função ST_GeomFromText. A primeira não pega nenhuma SRID e retorna uma geometria com um sistema de referência espacial indefinido (SRID=0). A segunda pega uma SRID como o segundo argumento e retorna uma geometria que inclui essa SRID como parte dos seus metadados.

This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2 - opção SRID é da suíte de conformidade.

This method implements the SQL/MM specification. SQL-MM 3: 5.1.40

This method supports Circular Strings and Curves.

Note
While not OGC-compliant, ST_MakePoint is faster than ST_GeomFromText and ST_PointFromText. It is also easier to use for numeric coordinate values. ST_Point is another option similar in speed to ST_MakePoint and is OGC-compliant, but doesn't support anything but 2D points.

Warning
Alterações: 2.0.0 Nas primeiras versões do PostGIS, ST_GeomFromText('GEOMETRYCOLLECTION(EMPTY)') foi permitida. Ela agora é ilegal no PostGIS 2.0.0 para melhor se adequar aos padrões SQL/MM. Ela deverá se escrita como ST_GeomFromText('GEOMETRYCOLLECTION EMPTY')

Exemplos

```
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)');
```

Veja também.

ST_GeomFromEWKT, ST_GeomFromWKB, ST_SRID

7.8.1.10 ST_LineFromText

ST_LineFromText — Faz uma geometria de uma representação WKT com a SRID dada. Se a SRID não for dada, isso leva a 0.

Synopsis

geometry **ST_LineFromText**(text WKT);
geometry **ST_LineFromText**(text WKT, integer srid);

Descrição

Makes a Geometry from WKT with the given SRID. If SRID is not given, it defaults to 0. If WKT passed in is not a LINESTRING, then null is returned.

Note
OGC SPEC 3.2.6.2 - opção SRID é de uma suíte de conformação.

Note
Se você sabe que todas as suas geometrias são LINESTRINGS, é mais eficiente usar somente ST_GeomFromText. Isso só convida a ST_GeomFromText e adiciona validação extra que ela retorna uma linestring.

This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2

This method implements the SQL/MM specification. SQL-MM 3: 7.2.8

Exemplos

```
SELECT ST_LineFromText('LINESTRING(1 2, 3 4)') AS aline, ST_LineFromText('POINT(1 2)') AS  ↩
    null_return;
aline                              | null_return
-----------------------------------------------
0102000000020000000000000000F ... | t
```

Veja também.

ST_GeomFromText

7.8.1.11 ST_MLineFromText

ST_MLineFromText — Retorna um valor específico ST_MultiLineString de uma representação WKT.

Synopsis

geometry **ST_MLineFromText**(text WKT, integer srid);
geometry **ST_MLineFromText**(text WKT);

Descrição

Makes a Geometry from Well-Known-Text (WKT) with the given SRID. If SRID is not given, it defaults to 0.

OGC SPEC 3.2.6.2 - opção SRID é de uma suíte de conformação

Retorna nulo se o WKT não é uma MULTILIINESTRING

Note
Se você tem total certeza de que todas suas geometrias WKT são pontos, não use essa função. Ela é mais devagar que a ST_GeomFromText, já que adiciona um passo de validação adicional.

This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2

This method implements the SQL/MM specification. SQL-MM 3: 9.4.4

Exemplos

```
SELECT ST_MLineFromText('MULTILINESTRING((1 2, 3 4), (4 5, 6 7))');
```

Veja também.

ST_GeomFromText

7.8.1.12 ST_MPointFromText

ST_MPointFromText — Makes a Geometry from WKT with the given SRID. If SRID is not given, it defaults to 0.

Synopsis

geometry **ST_MPointFromText**(text WKT, integer srid);
geometry **ST_MPointFromText**(text WKT);

Descrição

Makes a Geometry from WKT with the given SRID. If SRID is not given, it defaults to 0.

OGC SPEC 3.2.6.2 - opção SRID é de uma suíte de conformação

Retorna nulo se o WKT não é um MULTIPONTO

Note
Se você tem total certeza de que todas suas geometrias WKT são pontos, não use essa função. Ela é mais devagar que a ST_GeomFromText, já que adiciona um passo de validação adicional.

 This method implements the OGC Simple Features Implementation Specification for SQL 1.1. 3.2.6.2

 This method implements the SQL/MM specification. SQL-MM 3: 9.2.4

Exemplos

```
SELECT ST_MPointFromText('MULTIPOINT((1 2),(3 4))');
SELECT ST_MPointFromText('MULTIPOINT((-70.9590 42.1180),(-70.9611 42.1223))', 4326);
```

Veja também.

ST_GeomFromText

7.8.1.13 ST_MPolyFromText

ST_MPolyFromText — Makes a MultiPolygon Geometry from WKT with the given SRID. If SRID is not given, it defaults to 0.

Synopsis

geometry **ST_MPolyFromText**(text WKT, integer srid);
geometry **ST_MPolyFromText**(text WKT);

Descrição

Makes a MultiPolygon from WKT with the given SRID. If SRID is not given, it defaults to 0.

OGC SPEC 3.2.6.2 - opção SRID é de uma suíte de conformação

Descarta um erro se o WKT não for um MULTIPOLÍGONO

Note
Se você tem total certeza de que todas suas geometrias WKT são multipolígonos, não use essa função. Ela é mais devagar que a ST_GeomFromText, já que adiciona um passo de validação adicional.

 This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2

 This method implements the SQL/MM specification. SQL-MM 3: 9.6.4

Exemplos

```
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);
```

Veja também.

ST_GeomFromText, ST_SRID

7.8.1.14 ST_PointFromText

ST_PointFromText — Faz um ponto de um WKT com o SRID dado. Se o SRID não for dado, isso leva a desconhecido.

Synopsis

geometry **ST_PointFromText**(text WKT);
geometry **ST_PointFromText**(text WKT, integer srid);

Descrição

Constructs a PostGIS ST_Geometry point object from the OGC Well-Known text representation. If SRID is not given, it defaults to unknown (currently 0). If geometry is not a WKT point representation, returns null. If completely invalid WKT, then throws an error.

Note
Existem 2 variantes da função ST_PointFromText, a primeira não pega nenhuma SRID e retorna uma geometria sem sistema de referência espacial definido. A segunda, pega uma id referência espacial como o segundo argumento e retorna uma ST_Geometry que inclui esse srid como parte dos seus metadados. O srid deve ser definido na spatial_ref_sys table.

Note
Se você tem total certeza de que todas suas geometrias WKT são pontos, não use essa função. Ela é mais devagar que a ST_GeomFromText, já que adiciona um passo de validação adicional. Se você está construindo pontos de coordenadas long lat e se importa mais com apresentação e precisão do que com concordância OGC, use: ST_MakePoint ou ST_Point.

 This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2 - opção SRID é da suíte de conformidade.

 This method implements the SQL/MM specification. SQL-MM 3: 6.1.8

Exemplos

```
SELECT ST_PointFromText('POINT(-71.064544 42.28787)');
SELECT ST_PointFromText('POINT(-71.064544 42.28787)', 4326);
```

Veja também.

ST_GeomFromText, ST_MakePoint, ST_Point, ST_SRID

7.8.1.15 ST_PolygonFromText

ST_PolygonFromText — Makes a Geometry from WKT with the given SRID. If SRID is not given, it defaults to 0.

Synopsis

geometry **ST_PolygonFromText**(text WKT);
geometry **ST_PolygonFromText**(text WKT, integer srid);

Descrição

Makes a Geometry from WKT with the given SRID. If SRID is not given, it defaults to 0. Returns null if WKT is not a polygon.

OGC SPEC 3.2.6.2 - opção SRID é de uma suíte de conformação

Note
Se você tem total certeza de que todas suas geometrias WKT são polígonos, não use essa função. Ela é mais devagar que a ST_GeomFromText, já que adiciona um passo de validação adicional.

 This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2

 This method implements the SQL/MM specification. SQL-MM 3: 8.3.6

Exemplos

```
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
------------------
0103000000010000000050000006...


SELECT ST_PolygonFromText('POINT(1 2)') IS NULL as point_is_notpoly;

point_is_not_poly
----------
t
```

Veja também.

ST_GeomFromText

7.8.1.16 ST_WKTToSQL

ST_WKTToSQL — Retorna um valor ST_Geometry especifico da representação de texto estendida bem conhecida (EWKT). Isso é um heterônimo para ST_GeomFromText

Synopsis

geometria **ST_WKTToSQL**(texto WKT);

Descrição

 This method implements the SQL/MM specification. SQL-MM 3: 5.1.34

Veja também.

ST_GeomFromText

7.8.2 Well-Known Binary (WKB)

7.8.2.1 ST_GeogFromWKB

ST_GeogFromWKB — Cria uma ocasião geografia de uma geometria binária bem conhecida (WKB) ou binário estendido bem conhecido (EWKB).

Synopsis

geography **ST_GeogFromWKB**(bytea wkb);

Descrição

A função `ST_GeogFromWKB`, pega uma representação binária bem conhecida (WKB) de uma geometria ou WKB estendida do POstGIS e cria uma ocasião do tipo de geografia apropriado. Essa função cumpre o papel da Fábrica de Geometria em SQL.

Se a SRID não está especificado, isso leva a 4326 (WGS 84 long lat).

 This method supports Circular Strings and Curves.

Exemplos

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

Veja também.

ST_GeogFromText, ST_AsBinary

7.8.2.2 ST_GeomFromEWKB

ST_GeomFromEWKB — Retorna um valor ST_Geometry especifico da representação binária estendida bem conhecida (EWKB).

Synopsis

geometria **ST_GeomFromEWKB**(bytea EWKB);

Descrição

Constrói um objeto PostGIS ST_Geometry da representação binária estendida bem conhecida OGC (EWKT).

Note
O formato EWKB não é um padrão OGC, mas um formato específico PostGIS que inclui o identificador de sistema de referência espacial (SRID).

Melhorias: 2.0.0 suporte para superfícies poliédricas e TIN foi introduzido.

 This function supports 3d and will not drop the z-index.

 This method supports Circular Strings and Curves.

 This function supports Polyhedral surfaces.

 This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

Exemplos

line string binary rep 0f LINESTRING(-71.160281 42.258729,-71.160837 42.259113,-71.161144 42.25932) in NAD 83 long lat (4269).

Note
NOTA: Mesmo que os byte arrays seja delimitados com \ e talvez tenham ', precisamos escapar ambos com \ e " se as standard_conforming_strings estão deligadas. então, isso não parece exatamente como uma representação AsEWKB.

```
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@');
```

Note
In PostgreSQL 9.1+ - standard_conforming_strings is set to on by default, where as in past versions it was set to off. You can change defaults as needed for a single query or at the database or server level. Below is how you would do it with standard_conforming_strings = on. In this case we escape the ' with standard ansi ', but slashes are not escaped

```
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')
```

Veja também.

ST_AsBinary, ST_AsEWKB, ST_GeomFromWKB

7.8.2.3 ST_GeomFromWKB

ST_GeomFromWKB — Criar uma geometria exemplo de um representação bem conhecida de geometria binária (WKB) e SRID opcional.

Synopsis

geometry **ST_GeomFromWKB**(bytea geom);
geometry **ST_GeomFromWKB**(bytea geom, integer srid);

Descrição

A função `ST_GeomFromWKB`, pega uma representação binária bem conhecida de uma geometria e um sistema de referência espacial ID (`SRID`) e cria um exemplo do tipo apropriado de geometria. Essa função cumpre o papel da Fábrica de Geometria na SQL. Isso é um nome alternativo para ST_WKBToSQL.

Se o SRID não for especificado, leva a 0 (desconhecido).

This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s3.2.7.2 - o SRID opcional é da suíte de conformidade.

This method implements the SQL/MM specification. SQL-MM 3: 5.1.41

This method supports Circular Strings and Curves.

Exemplos

```
--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)
```

Veja também.

ST_WKBToSQL, ST_AsBinary, ST_GeomFromEWKB

7.8.2.4 ST_LineFromWKB

ST_LineFromWKB — Faz uma `LINESTRING` de uma WKB com o SRID dado

Synopsis

geometry **ST_LineFromWKB**(bytea WKB);
geometry **ST_LineFromWKB**(bytea WKB, integer srid);

Descrição

A função `ST_LineFromWKB`, pega uma representação binária bem conhecida de geometria e um sistema de referência espacial ID (`SRID`) e cria um exemplo do tipo apropriado de geometria - nesse caso, uma geometria `LINESTRING` . Essa função cumpre o papel da Fábrica de Geometria SQL.

Se um SRID não estiver especificado, isso leva a 0. Retorna `NULA` se a entrada `bytea` não representa uma `LINESTRING`.

Note
OGC SPEC 3.2.6.2 - opção SRID é de uma suíte de conformação.

Note
Se você sabe que todas suas geometrias são `LINESTRING`s, é mais eficaz usar ST_GeomFromWKB. Essa função convida ST_GeomFromWKB e adiciona validação extra que ela retorna uma linestring.

This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2

This method implements the SQL/MM specification. SQL-MM 3: 7.2.9

Exemplos

```
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
-----------------------------------------------
0102000000020000000000000000F ... | t
```

Veja também.

ST_GeomFromWKB, ST_LinestringFromWKB

7.8.2.5 ST_LinestringFromWKB

ST_LinestringFromWKB — Faz uma geometria de uma WKB com o SRID dado.

Synopsis

geometry **ST_LinestringFromWKB**(bytea WKB);
geometry **ST_LinestringFromWKB**(bytea WKB, integer srid);

Descrição

A função `ST_LinestringFromWKB`, pega uma representação binária bem conhecida de geometria e um sistema de referência espacial ID (`SRID`) e cria um exemplo do tipo apropriado de geometria - nesse caso, uma geometria `LINESTRING` . Essa função cumpre o papel da Fábrica de Geometria SQL.

Se um SRID não estiver especificado, isso leva a 0. Retorna `NULA` se a entrada `bytea` não representa uma geometria `LINESTRING`. Isso é um heterônimo para ST_LineFromWKB.

Note
OGC SPEC 3.2.6.2 - o SRID opcional é da suíte de conformação.

Note
Se você sabe que todas suas geometrias são `LINESTRING`s, é mais eficaz usar ST_GeomFromWKB. Essa função convida ST_GeomFromWKB e adiciona validação extra que ela retorna uma `LINESTRING`.

This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2

This method implements the SQL/MM specification. SQL-MM 3: 7.2.9

Exemplos

```
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
-----------------------------------------------
0102000000020000000000000000F ... | t
```

Veja também.

ST_GeomFromWKB, ST_LineFromWKB

7.8.2.6 ST_PointFromWKB

ST_PointFromWKB — Faz uma geometria a partir de um WKB com o SRID dado

Synopsis

geometry **ST_GeomFromWKB**(bytea geom);
geometry **ST_GeomFromWKB**(bytea geom, integer srid);

Descrição

A função `ST_PointFromWKB`, pega uma representação binária bem conhecida de geometria e um sistema de referência espacial ID (`SRID`) e cria um exemplo do tipo apropriado de geometria - nesse caso, uma geometria `PONTO` . Essa função cumpre o papel da Fábrica de Geometria SQL.

Se uma SRID não for especificada, leva a 0. `NULO` é retornado se a entrada `bytea` não representar uma `PONTO` geometria.

This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s3.2.7.2

This method implements the SQL/MM specification. SQL-MM 3: 6.1.9

This function supports 3d and will not drop the z-index.

This method supports Circular Strings and Curves.

Exemplos

```
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)
```

Veja também.

ST_GeomFromWKB, ST_LineFromWKB

7.8.2.7 ST_WKBToSQL

ST_WKBToSQL — Retorna um valor ST_Geometry especifico da representação de texto binário bem conhecida (WKB). Isso é um heterônimo para ST_GeomFromWKB que não pega nenhum srid

Synopsis

geometry **ST_WKBToSQL**(bytea WKB);

Descrição

This method implements the SQL/MM specification. SQL-MM 3: 5.1.36

Veja também.

ST_GeomFromWKB

7.8.3 Other Formats

7.8.3.1 ST_Box2dFromGeoHash

ST_Box2dFromGeoHash — Retorna uma CAIXA2D de uma string GeoHash.

Synopsis

box2d **ST_Box2dFromGeoHash**(text geohash, integer precision=full_precision_of_geohash);

Descrição

Retorna uma CAIXA2D de uma string GeoHash.

If no `precision` is specified ST_Box2dFromGeoHash returns a BOX2D based on full precision of the input GeoHash string.

Se a `precisão` é especificada, a ST_Box2dFromGeoHash irá usar aqueles vários caracteres do GeoHash para criar a CAIXA2D. Valores de precisão mais baixos resultam em CAIXAS2D maiores e valores maiores aumentam a precisão.

Disponibilidade: 2.1.0

Exemplos

```
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)
```

Veja também.

ST_GeoHash, ST_GeomFromGeoHash, ST_PointFromGeoHash

7.8.3.2 ST_GeomFromGeoHash

ST_GeomFromGeoHash — Retorna uma geometria de uma string GeoHash.

Synopsis

geometria **ST_GeomFromGeoHash**(texto geohash, inteiro precision=full_precision_of_geohash);

Descrição

Retorna uma geometria de uma string GeoHash. A geometria será um polígono representando os limites GeoHash.

Se nenhuma `precisão` for especificada, a ST_GeomFromGeoHash retorna um polígono baseado na precisão completa da string de entrada GeoHash.

Se a `precisão` for especificada, a ST_GeomFromGeoHash irá usar aqueles vários caracteres do GeoHash para criar o polígono.

Disponibilidade: 2.1.0

Exemplos

```
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))
```

Veja também.

ST_GeoHash,ST_Box2dFromGeoHash, ST_PointFromGeoHash

7.8.3.3 ST_GeomFromGML

ST_GeomFromGML — Utiliza como entrada uma representação GML de geometria e como saída um objeto de geometria PostGIS

Synopsis

geometry **ST_GeomFromGML**(text geomgml);
geometry **ST_GeomFromGML**(text geomgml, integer srid);

Descrição

Constrói um objeto PostGIS ST_Geometry de uma representação OGC GML.

A ST_GeomFromGML funciona apenas para fragmentos da geometria GML. Ela descarta um erro, se você tentar usá-la em um documento inteiro GML.

OGC GML versions supported:

- GML 3.2.1 Namespace
- GML 3.1.1 Simple Features profile SF-2 (with GML 3.1.0 and 3.0.0 backward compatibility)
- GML 2.1.2

OGC GML standards, cf: http://www.opengeospatial.org/standards/gml:

Disponibilidade:1.5, requer libxml2 1.6+

Melhorias: 2.0.0 suporte para superfícies poliédricas e TIN foi introduzido.

Melhorias: 2.0.0 parâmetro opcional padrão srid adicionado.

 This function supports 3d and will not drop the z-index.

 This function supports Polyhedral surfaces.

 This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

GML permite dimensões mescladas (2D e 3D dentro da mesma MultiGeometria, por exemplo). ST_GeomFromGML converte a geometria inteira para 2D se uma dimensão Z perdida for encontrada uma vez, as geometrias do PostGIS não fazem isso.

A GML suporta SRS misturadas dentro da mesma MultiGeometria. As ST_GeomFromGML, reprojetam todas as sub geometrias para o nó raiz da SRS, as geometrias PostGIS não fazem isso. Se nenhum srsNome atribui disponível para o nó raiz GML, a função descarta um erro.

A função ST_GeomFromGML não é afetada sobre um espaço de nome específico GML. Você poderia evitar mencionar ela explicitamente para usos comuns. Mas você precisa dela se quiser usar XLink dentro de GML.

Note
A função ST_GeomFromGML não suporta geometrias SQL/MM curvas.

Exemplos - Uma única geometria com srsNome

```
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
>');
```

Exemplos - Uso XLink

```
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
>'););
```

Exemplos - Superfícies Poliédricas

```
SELECT ST_AsEWKT(ST_GeomFromGML('
<gml:PolyhedralSurface>
<gml:polygonPatches>
  <gml:PolygonPatch>
    <gml:exterior>
      <gml:LinearRing
><gml:posList srsDimension="3"
>0 0 0 0 0 1 0 1 1 0 1 0 0 0 0</gml:posList
></gml:LinearRing>
    </gml:exterior>
  </gml:PolygonPatch>
```

```
  <gml:PolygonPatch>
    <gml:exterior>
        <gml:LinearRing
><gml:posList srsDimension="3"
>0 0 0 0 1 0 1 1 0 1 0 0 0 0 0</gml:posList
></gml:LinearRing>
    </gml:exterior>
  </gml:PolygonPatch>
  <gml:PolygonPatch>
    <gml:exterior>
        <gml:LinearRing
><gml:posList srsDimension="3"
>0 0 0 1 0 0 1 0 1 0 0 1 0 0 0</gml:posList
></gml:LinearRing>
    </gml:exterior>
  </gml:PolygonPatch>
  <gml:PolygonPatch>
    <gml:exterior>
        <gml:LinearRing
><gml:posList srsDimension="3"
>1 1 0 1 1 1 1 0 1 1 0 0 1 1 0</gml:posList
></gml:LinearRing>
    </gml:exterior>
  </gml:PolygonPatch>
  <gml:PolygonPatch>
    <gml:exterior>
        <gml:LinearRing
><gml:posList srsDimension="3"
>0 1 0 0 1 1 1 1 1 1 1 0 0 1 0</gml:posList
></gml:LinearRing>
    </gml:exterior>
  </gml:PolygonPatch>
  <gml:PolygonPatch>
    <gml:exterior>
        <gml:LinearRing
><gml:posList srsDimension="3"
>0 0 1 1 0 1 1 1 1 0 1 1 0 0 1</gml: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 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)))
```

Veja também.

Section 2.2.3, ST_AsGML, ST_GMLToSQL

7.8.3.4 ST_GeomFromGeoJSON

ST_GeomFromGeoJSON — Utiliza como entrada uma representação geojson de uma geometria e como saída um objeto de geometria PostGIS

Synopsis

geometry **ST_GeomFromGeoJSON**(text geomjson);
geometry **ST_GeomFromGeoJSON**(json geomjson);
geometry **ST_GeomFromGeoJSON**(jsonb geomjson);

Descrição

Constrói um objeto de geometria PostGIS de uma representação GeoJSON.

A ST_GeomFromGeoJSON funciona apenas para fragmentos da geometria JSON. Ela descarta um erro se você tentar usá-la em um documento JSON inteiro.

Enhanced: 3.0.0 parsed geometry defaults to SRID=4326 if not specified otherwise.

Enhanced: 2.5.0 can now accept json and jsonb as inputs.

Disponibilidade: 2.0.0 requer - JSON-C >= 0.9

Note
Se você não tem JSON-C ativada, o suporte apresentará uma notificação de erro ao invés de uma saída. Para ativar JSON-C, execute a configuração --with-jsondir=/path/to/json-c. Veja mais detalhes em: Section 2.2.3.

 This function supports 3d and will not drop the z-index.

Exemplos

```
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)
```

Veja também.

ST_AsText, ST_AsGeoJSON, Section 2.2.3

7.8.3.5 ST_GeomFromKML

ST_GeomFromKML — Utiliza como entrada uma representação KML de geometria e como saída um objeto de geometria PostGIS

Synopsis

geometria **ST_GeomFromKML**(texto geomkml);

Descrição

Constrói um objeto PostGIS ST_Geometry de uma representação OGC KML.

A ST_GeomFromKML funciona apenas para fragmentos da geometria KML. Ela descarta um erro, se você tentar usá-la em um documento inteiro KML.

OGC KML versões suportadas:

- KML 2.2.0 Namespace

OGC KML standards, cf: http://www.opengeospatial.org/standards/kml:

Availability: 1.5, requires libxml2 2.6+

 This function supports 3d and will not drop the z-index.

Note
A função ST_GeomFromKML não suporta geometrias SQL/MM curvas.

Exemplos - Uma única geometria com srsNome

```
SELECT ST_GeomFromKML('
                <LineString>
                        <coordinates
>-71.1663,42.2614
                                -71.1667,42.2616</coordinates>
                </LineString
>');
```

Veja também.

Section 2.2.3, ST_AsKML

7.8.3.6 ST_GeomFromTWKB

ST_GeomFromTWKB — Cria uma ocasião de uma TWKB ("Tiny Well-Known Binary") representação de geometria.

Synopsis

geometria **ST_GeomFromTWKB**(bytea twkb);

Descrição

A função `ST_GeomFromTWKB`, pega uma TWKB ("Tiny Well-Known Binary") representação geométrica (WKB) e cria uma ocasião do tipo apropriado de geometria.

Exemplos

```
SELECT ST_AsText(ST_GeomFromTWKB(ST_AsTWKB('LINESTRING(126 34, 127 35)'::geometry)));

         st_astext
----------------------------
 LINESTRING(126 34, 127 35)
(1 row)


SELECT ST_AsEWKT(
  ST_GeomFromTWKB(E'\\x620002f7f40dbce4040105')
);
                                          st_asewkt
------------------------------------------------------
LINESTRING(-113.98 39.198,-113.981 39.195)
(1 row)
```

Veja também.

ST_AsTWKB

7.8.3.7 ST_GMLToSQL

ST_GMLToSQL — Retorna um valor ST_Geometry específico da representação GML. Esse é um heterônimo para ST_GeomFromGML

Synopsis

geometry **ST_GMLToSQL**(text geomgml);
geometry **ST_GMLToSQL**(text geomgml, integer srid);

Descrição

 This method implements the SQL/MM specification. SQL-MM 3: 5.1.50 (exceto para curvas suporte).

Disponibilidade:1.5, requer libxml2 1.6+

Melhorias: 2.0.0 suporte para superfícies poliédricas e TIN foi introduzido.

Melhorias: 2.0.0 parâmetro opcional padrão srid adicionado.

Veja também.

Section 2.2.3, ST_GeomFromGML, ST_AsGML

7.8.3.8 ST_LineFromEncodedPolyline

ST_LineFromEncodedPolyline — Cria uma LineString de uma Encoded Polyline.

Synopsis

geometria **ST_LineFromEncodedPolyline**(texto polyline, inteiro precision=5);

Descrição

Cria uma LineString de uma string Encoded Polyline.

Optional `precision` specifies how many decimal places will be preserved in Encoded Polyline. Value should be the same on encoding and decoding, or coordinates will be incorrect.

Veja http://developers.google.com/maps/documentation/utilities/polylinealgorithm

Disponibilidade: 2.2.0

Exemplos

```
-- Create a line string from a polyline
SELECT ST_AsEWKT(ST_LineFromEncodedPolyline('_p~iF~ps|U_ulLnnqC_mqNvxq`@'));
-- result --
SRID=4326;LINESTRING(-120.2 38.5,-120.95 40.7,-126.453 43.252)

-- Select different precision that was used for polyline encoding
SELECT ST_AsEWKT(ST_LineFromEncodedPolyline('_p~iF~ps|U_ulLnnqC_mqNvxq`@',6));
-- result --
SRID=4326;LINESTRING(-12.02 3.85,-12.095 4.07,-12.6453 4.3252)
```

Veja também.

ST_AsEncodedPolyline

7.8.3.9 ST_PointFromGeoHash

ST_PointFromGeoHash — Retorna um ponto de uma string GeoHash.

Synopsis

ponto **ST_PointFromGeoHash**(texto geohash, inteiro precision=full_precision_of_geohash);

Descrição

Retorna um ponto de uma string GeoHash. O ponto representa o ponto central do GeoHash.

Se nenhuma `precisão` for especificada, a ST_PointFromGeoHash retorna um ponto baseado na precisão completa da string da entrada GeoHash.

Se a `precisão` for especificada, a ST_PointFromGeoHash irá usar aqueles vários caracteres do GeoHash para criar o ponto.

Disponibilidade: 2.1.0

Exemplos

```
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)
```

Veja também.

ST_GeoHash, ST_Box2dFromGeoHash, ST_GeomFromGeoHash

7.8.3.10 ST_FromFlatGeobufToTable

ST_FromFlatGeobufToTable — Creates a table based on the structure of FlatGeobuf data.

Synopsis

geometria **ST_BdPolyFromText**(texto WKT, inteiro srid);

Descrição

Creates a table based on the structure of FlatGeobuf data. (http://flatgeobuf.org).

`schema` Schema name.

`table` Table name.

`data` Input FlatGeobuf data.

Availability: 3.2.0

7.8.3.11 ST_FromFlatGeobuf

ST_FromFlatGeobuf — Reads FlatGeobuf data.

Synopsis

setof anyelement **ST_FromFlatGeobuf**(anyelement Table reference, bytea FlatGeobuf input data);

Descrição

Reads FlatGeobuf data (http://flatgeobuf.org). NOTE: PostgreSQL bytea cannot exceed 1GB.

`tabletype` reference to a table type.

`data` input FlatGeobuf data.

Availability: 3.2.0

7.9 Geometry Output

7.9.1 Well-Known Text (WKT)

7.9.1.1 ST_AsEWKT

ST_AsEWKT — Retorna a representação de texto bem conhecida (WKT) da geometria com os meta dados SRID.

Synopsis

text **ST_AsEWKT**(geometry g1);
text **ST_AsEWKT**(geometry g1, integer maxdecimaldigits=15);
text **ST_AsEWKT**(geography g1);
text **ST_AsEWKT**(geography g1, integer maxdecimaldigits=15);

Descrição

Returns the Well-Known Text representation of the geometry prefixed with the SRID. The optional *maxdecimaldigits* argument may be used to reduce the maximum number of decimal digits after floating point used in output (defaults to 15).

To perform the inverse conversion of EWKT representation to PostGIS geometry use ST_GeomFromEWKT.

Warning
Using the *maxdecimaldigits* parameter can cause output geometry to become invalid. To avoid this use ST_ReducePrecision with a suitable gridsize first.

Note
The WKT spec does not include the SRID. To get the OGC WKT format use ST_AsText.

Warning
WKT format does not maintain precision so to prevent floating truncation, use ST_AsBinary or ST_AsEWKB format for transport.

Enhanced: 3.1.0 support for optional precision parameter.

Melhorias: 2.0.0 suporte para geografia, superfícies poliédricas, triângulos e TIN foi introduzido.


This function supports 3d and will not drop the z-index.


This method supports Circular Strings and Curves.


This function supports Polyhedral surfaces.


This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

Examples

```
SELECT ST_AsEWKT('0103000020E6100000010000005000000000000000
                        0000000000000000000000000000000000000000000000000000000000
                        F03F000000000000F03F000000000000F03F000000000000F03
                        F0000000000000000000000000000000000000000000000000'::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)
```

Veja também.

ST_AsBinary, ST_AsEWKB, ST_AsText, ST_GeomFromEWKT

7.9.1.2 ST_AsText

ST_AsText — Retorna a representação de texto bem conhecida (WKT) da geometria/geografia sem os meta dados do SRID.

Synopsis

text **ST_AsText**(geometry g1);
text **ST_AsText**(geometry g1, integer maxdecimaldigits = 15);
text **ST_AsText**(geography g1);
text **ST_AsText**(geography g1, integer maxdecimaldigits = 15);

Descrição

Returns the OGC Well-Known Text (WKT) representation of the geometry/geography. The optional *maxdecimaldigits* argument may be used to limit the number of digits after the decimal point in output ordinates (defaults to 15).

To perform the inverse conversion of WKT representation to PostGIS geometry use ST_GeomFromText.

Note
The standard OGC WKT representation does not include the SRID. To include the SRID as part of the output representation, use the non-standard PostGIS function ST_AsEWKT

Warning
The textual representation of numbers in WKT may not maintain full floating-point precision. To ensure full accuracy for data storage or transport it is best to use Well-Known Binary (WKB) format (see ST_AsBinary and *maxdecimaldigits*).

Warning
Using the *maxdecimaldigits* parameter can cause output geometry to become invalid. To avoid this use ST_ReducePrecision with a suitable gridsize first.

Disponibilidade: 1.5 - suporte para geografia foi introduzido.

Enhanced: 2.5 - optional parameter precision introduced.


This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.1


This method implements the SQL/MM specification. SQL-MM 3: 5.1.25


This method supports Circular Strings and Curves.

Examples

```
SELECT ST_AsText('0103000000010000000500000000000000000000
0000000000000000000000000000000000000000000000000000
F03F000000000000F03F000000000000F03F000000000000F03
F000000000000000000000000000000000000000000000000000');

    st_astext
------------------------------
 POLYGON((0 0,0 1,1 1,1 0,0 0))
```

Full precision output is the default.

```
SELECT ST_AsText('POINT(111.1111111 1.1111111)'));
    st_astext
------------------------------
 POINT(111.1111111 1.1111111)
```

The *maxdecimaldigits* argument can be used to limit output precision.

```
SELECT ST_AsText('POINT(111.1111111 1.1111111)'), 2);
    st_astext
--------------------
 POINT(111.11 1.11)
```

Veja também.

ST_AsBinary, ST_AsEWKB, ST_AsEWKT, ST_GeomFromText

7.9.2 Well-Known Binary (WKB)

7.9.2.1 ST_AsBinary

ST_AsBinary — Return the OGC/ISO Well-Known Binary (WKB) representation of the geometry/geography without SRID meta data.

Synopsis

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);

Descrição

Returns the OGC/ISO Well-Known Binary (WKB) representation of the geometry. The first function variant defaults to encoding using server machine endian. The second function variant takes a text argument specifying the endian encoding, either little-endian ('NDR') or big-endian ('XDR').

WKB format is useful to read geometry data from the database and maintaining full numeric precision. This avoids the precision rounding that can happen with text formats such as WKT.

To perform the inverse conversion of WKB to PostGIS geometry use ST_GeomFromWKB.

Note
The OGC/ISO WKB format does not include the SRID. To get the EWKB format which does include the SRID use ST_AsEWKB

> **Note**
> The default behavior in PostgreSQL 9.0 has been changed to output bytea in hex encoding. If your GUI tools require the old behavior, then SET bytea_output='escape' in your database.

Melhorias: 2.0.0 suporte para superfícies poliédricas, triângulos e TINs introduzido.

Melhorias: 2.0.0 suporte para maiores dimensões de coordenadas foi introduzido.

Melhorias: 2.0.0 suporte para edian especificando com geografia foi introduzido.

Disponibilidade: 1.5.0 suporte para geografia foi introduzido.

Alterações: 2.0.0 Entrada para esta função não pode ser desconhecida -- deve ser geometria. Construções como `ST_AsBinary('POINT(1 2)')` não são mais válidas e você terá `n st_asbinary(desconhecido) não é um erro único`. Códigos assim, precisam ser alterados para `ST_AsBinary('POINT(1 2)'::geometry);`. Se não for possível, instale: `legacy.sql`.

This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.1

This method implements the SQL/MM specification. SQL-MM 3: 5.1.37

This method supports Circular Strings and Curves.

This function supports Polyhedral surfaces.

This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

This function supports 3d and will not drop the z-index.

Examples

```
SELECT ST_AsBinary(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))',4326));

                   st_asbinary
--------------------------------
\x01030000000100000005000000000000000000000000000000000000000000000000000000
000000f03f000000000000f03f000000000000f03f000000000000f03f000000000000000000
00000000000000000000000000
```

```
SELECT ST_AsBinary(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))',4326), 'XDR');
                   st_asbinary
--------------------------------
\x000000000300000001000000050000000000000000000000000000000000000000003ff000
00000000003ff00000000000003ff00000000000003ff0000000000000000000000000000000
00000000000000000000000000
```

Veja também.

ST_GeomFromWKB, ST_AsEWKB, ST_AsTWKB, ST_AsText,

7.9.2.2 ST_AsEWKB

ST_AsEWKB — Return the Extended Well-Known Binary (EWKB) representation of the geometry with SRID meta data.

Synopsis

bytea **ST_AsEWKB**(geometry g1);
bytea **ST_AsEWKB**(geometry g1, text NDR_or_XDR);

Descrição

Returns the Extended Well-Known Binary (EWKB) representation of the geometry with SRID metadata. The first function variant defaults to encoding using server machine endian. The second function variant takes a text argument specifying the endian encoding, either little-endian ('NDR') or big-endian ('XDR').

WKB format is useful to read geometry data from the database and maintaining full numeric precision. This avoids the precision rounding that can happen with text formats such as WKT.

To perform the inverse conversion of EWKB to PostGIS geometry use ST_GeomFromEWKB.

Note
To get the OGC/ISO WKB format use ST_AsBinary. Note that OGC/ISO WKB format does not include the SRID.

Melhorias: 2.0.0 suporte para superfícies poliédricas, triângulos e TINs introduzido.


This function supports 3d and will not drop the z-index.


This method supports Circular Strings and Curves.


This function supports Polyhedral surfaces.


This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

Examples

```
SELECT ST_AsEWKB(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))',4326));

                   st_asewkb
---------------------------------
\x0103000020e6100000010000000500000000000000000000000000000000000000000000000000000000000000
0000000000000f03f000000000000f03f000000000000f03f000000000000f03f000000000000000
00000000000000000000000000000000
```

```
SELECT ST_AsEWKB(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))',4326), 'XDR');
                   st_asewkb
---------------------------------
\x0020000003000010e600000001000000050000000000000000000000000000000000000000000000000000000000
003ff0000000000003ff0000000000003ff0000000000003ff000000000000000000000000000
00000000000000000000000000000000
```

Veja também.

ST_AsBinary, ST_GeomFromEWKB, ST_SRID

7.9.2.3 ST_AsHEXEWKB

ST_AsHEXEWKB — Retorna uma geometria no formato HEXEWKB (como texto) usando little-endian (NDR) ou big-endian (XDR) encoding.

Synopsis

text **ST_AsHEXEWKB**(geometry g1, text NDRorXDR);
text **ST_AsHEXEWKB**(geometry g1);

Descrição

Retorna uma geometria no formato HEXEWKB (como texto) usando little-endian (NDR) ou big-endian (XDR) encoding. Se nenhum encoding estiver especificado, então o NDR é usado.

Note
Disponibilidade: 1.2.2

 This function supports 3d and will not drop the z-index.

 This method supports Circular Strings and Curves.

Examples

```
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
                0000000000000000000000000000000000
                0000000000000000000000000000000F03F
                000000000000F03F000000000000F03F000000000000F03
                F0000000000000000000000000000000000000000000000000
```

7.9.3 Other Formats

7.9.3.1 ST_AsEncodedPolyline

ST_AsEncodedPolyline — Retorna uma Polilinha Encoded de uma geometria LineString.

Synopsis

text **ST_AsEncodedPolyline**(geometry geom, integer precision=5);

Descrição

Returns the geometry as an Encoded Polyline. This format is used by Google Maps with precision=5 and by Open Source Routing Machine with precision=5 and 6.

Optional `precision` specifies how many decimal places will be preserved in Encoded Polyline. Value should be the same on encoding and decoding, or coordinates will be incorrect.

Disponibilidade: 2.2.0

Examples

Básico

```
SELECT ST_AsEncodedPolyline(GeomFromEWKT('SRID=4326;LINESTRING(-120.2 38.5,-120.95 ↩
    40.7,-126.453 43.252)'));
        --result--
        |_p~iF~ps|U_ulLnnqC_mqNvxq`@
```

Use em conjunto com a linestring geografia e segmentize geografia, e coloque no google maps

```
-- the SQL for Boston to San Francisco, segments every 100 KM
        SELECT ST_AsEncodedPolyline(
                ST_Segmentize(
                        ST_GeogFromText('LINESTRING(-71.0519 42.4935,-122.4483 37.64)'),
                                100000)::geometry) As encodedFlightPath;
```

javascript irá parecer em algo com isso, onde a variável $ você substitui com o resultado da pesquisa

```
<script type="text/javascript" src="http://maps.googleapis.com/maps/api/js?libraries= ↩
    geometry"
></script>
<script type="text/javascript">
         flightPath = new google.maps.Polyline({
                        path:  google.maps.geometry.encoding.decodePath("$encodedFlightPath ↩
                            "),
                        map: map,
                        strokeColor: '#0000CC',
                        strokeOpacity: 1.0,
                        strokeWeight: 4
                });
</script>
```

Veja também.

ST_LineFromEncodedPolyline, ST_Segmentize

7.9.3.2 ST_AsFlatGeobuf

ST_AsFlatGeobuf — Return a FlatGeobuf representation of a set of rows.

Synopsis

bytea **ST_AsFlatGeobuf**(anyelement set row);
bytea **ST_AsFlatGeobuf**(anyelement row, bool index);
bytea **ST_AsFlatGeobuf**(anyelement row, bool index, text geom_name);

Descrição

Return a FlatGeobuf representation (http://flatgeobuf.org) of a set of rows corresponding to a FeatureCollection. NOTE: PostgreSQL bytea cannot exceed 1GB.

`row` row data with at least a geometry column.

`index` toggle spatial index creation. Default is false.

`geom_name` is the name of the geometry column in the row data. If NULL it will default to the first found geometry column.

Availability: 3.2.0

7.9.3.3 ST_AsGeobuf

ST_AsGeobuf — Return a Geobuf representation of a set of rows.

Synopsis

bytea **ST_AsGeobuf**(anyelement set row);
bytea **ST_AsGeobuf**(anyelement row, text geom_name);

Descrição

Return a Geobuf representation (https://github.com/mapbox/geobuf) of a set of rows corresponding to a FeatureCollection. Every input geometry is analyzed to determine maximum precision for optimal storage. Note that Geobuf in its current form cannot be streamed so the full output will be assembled in memory.

`row` row data with at least a geometry column.

`geom_name` is the name of the geometry column in the row data. If NULL it will default to the first found geometry column.

Availability: 2.4.0

Examples

```
SELECT encode(ST_AsGeobuf(q, 'geom'), 'base64')
    FROM (SELECT ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))') AS geom) AS q;
 st_asgeobuf
---------------------------------
 GAAiEAoOCgwIBBoIAAAAAgIAAAE=
```

7.9.3.4 ST_AsGeoJSON

ST_AsGeoJSON — Return a geometry as a GeoJSON element.

Synopsis

text **ST_AsGeoJSON**(record feature, text geomcolumnname, integer maxdecimaldigits=9, boolean pretty_bool=false);
text **ST_AsGeoJSON**(geometry geom, integer maxdecimaldigits=9, integer options=8);
text **ST_AsGeoJSON**(geography geog, integer maxdecimaldigits=9, integer options=0);

Descrição

Returns a geometry as a GeoJSON "geometry", or a row as a GeoJSON "feature". (See the GeoJSON specifications RFC 7946). 2D and 3D Geometries are both supported. GeoJSON only support SFS 1.1 geometry types (no curve support for example).

The `maxdecimaldigits` argument may be used to reduce the maximum number of decimal places used in output (defaults to 9). If you are using EPSG:4326 and are outputting the geometry only for display, `maxdecimaldigits`=6 can be a good choice for many maps.

Warning
Using the *`maxdecimaldigits`* parameter can cause output geometry to become invalid. To avoid this use ST_ReducePrecision with a suitable gridsize first.

The `options` argument can be used to add BBOX or CRS in GeoJSON output:

- 0: means no option
- 1: GeoJSON BBOX
- 2: GeoJSON Short CRS (e.g EPSG:4326)
- 4: GeoJSON Long CRS (e.g urn:ogc:def:crs:EPSG::4326)
- 8: GeoJSON Short CRS if not EPSG:4326 (default)

The GeoJSON specification states that polygons are oriented using the Right-Hand Rule, and some clients require this orientation. This can be ensured by using ST_ForcePolygonCCW . The specification also requires that geometry be in the WGS84 coordinate system (SRID = 4326). If necessary geometry can be projected into WGS84 using ST_Transform: `ST_Transform( geom, 4326 )`.

GeoJSON can be tested and viewed online at geojson.io and geojsonlint.com. It is widely supported by web mapping frameworks:

- OpenLayers GeoJSON Example
- Leaflet GeoJSON Example
- Mapbox GL GeoJSON Example

Disponibilidade: 1.3.4

Disponibilidade: 1.5.0 suporte para geografia foi introduzido.

Alterações: 2.0.0 suporte padrão args e args nomeados.

Changed: 3.0.0 support records as input

Changed: 3.0.0 output SRID if not EPSG:4326.

 This function supports 3d and will not drop the z-index.

Examples

Generate a FeatureCollection:

```
SELECT json_build_object(
    'type', 'FeatureCollection',
    'features', json_agg(ST_AsGeoJSON(t.*)::json)
    )
FROM ( VALUES (1, 'one', 'POINT(1 1)'::geometry),
              (2, 'two', 'POINT(2 2)'),
              (3, 'three', 'POINT(3 3)')
     ) as t(id, name, geom);
```

```
{"type" : "FeatureCollection", "features" : [{"type": "Feature", "geometry": {"type":"Point ↩
    ","coordinates":[1,1]}, "properties": {"id": 1, "name": "one"}}, {"type": "Feature", " ↩
    geometry": {"type":"Point","coordinates":[2,2]}, "properties": {"id": 2, "name": "two ↩
    "}}, {"type": "Feature", "geometry": {"type":"Point","coordinates":[3,3]}, "properties": ↩
     {"id": 3, "name": "three"}}]}
```

Generate a Feature:

```
SELECT ST_AsGeoJSON(t.*)
FROM (VALUES (1, 'one', 'POINT(1 1)'::geometry)) AS t(id, name, geom);
```

```
st_asgeojson
-----------------------------------------------------------------------------------------------------------

 {"type": "Feature", "geometry": {"type":"Point","coordinates":[1,1]}, "properties": {"id": ↩
     1, "name": "one"}}
```

An alternate way to generate Features with an `id` property is to use JSONB functions and operators:

```
SELECT jsonb_build_object(
    'type',       'Feature',
    'id',         id,
    'geometry',   ST_AsGeoJSON(geom)::jsonb,
    'properties', to_jsonb( t.* ) - 'id' - 'geom'
    ) AS json
FROM (VALUES (1, 'one', 'POINT(1 1)'::geometry)) AS t(id, name, geom);
```

```
json
-----------------------------------------------------------------------------------------------------------

 {"id": 1, "type": "Feature", "geometry": {"type": "Point", "coordinates": [1, 1]}, " ↩
    properties": {"name": "one"}}
```

Don't forget to transform your data to WGS84 longitude, latitude to conform with the GeoJSON specification:

```
SELECT ST_AsGeoJSON(ST_Transform(geom,4326)) from fe_edges limit 1;
```

```
st_asgeojson
-----------------------------------------------------------------------------------------------------------


{"type":"MultiLineString","coordinates":[[[-89.734634999999997,31.492072000000000],
[-89.734955999999997,31.492237999999997]]]}
```

3D geometries are supported:

```
SELECT ST_AsGeoJSON('LINESTRING(1 2 3, 4 5 6)');
```

```
{"type":"LineString","coordinates":[[1,2,3],[4,5,6]]}
```

Veja também.

ST_GeomFromGeoJSON, ST_ForcePolygonCCW , ST_Transform

7.9.3.5 ST_AsGML

ST_AsGML — Retorna a geometria como uma versão GML com 2 ou 3 elementos.

Synopsis

text **ST_AsGML**(geometry geom, integer maxdecimaldigits=15, integer options=0);
text **ST_AsGML**(geography geog, integer maxdecimaldigits=15, integer options=0, text nprefix=null, text id=null);
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);

Descrição

Return the geometry as a Geography Markup Language (GML) element. The version parameter, if specified, may be either 2 or 3. If no version parameter is specified then the default is assumed to be 2. The `maxdecimaldigits` argument may be used to reduce the maximum number of decimal places used in output (defaults to 15).

Warning
Using the *`maxdecimaldigits`* parameter can cause output geometry to become invalid. To avoid this use ST_ReducePrecision with a suitable gridsize first.

GML 2 refere-se a versão 2.1.2 , GML 3 para a versão 3.1.1

O argumento "opções" é um bitfield. Ele poderia ser usado para definir o tipo de saída CRS na saída GML, e para declarar dados como lat/lon:

- 0: GML Short CRS (ex: EPSG:4326), valor padrão
- 1: GML Long CRS (ex: urn:ogc:def:crs:EPSG::4326)
- 2: Para GML 3 somente, remove srsDimension atribuída da saída.
- 4: Para GML 3 somente, use <LineString> em vez de <Curve> tag para linhas.
- 16: Declara que dados são lat/lon (ex: srid=4326). O padrão é supor que os dados são planos. Esta opção é útil apenas para saída GML 3.1.1, relacionada a ordem do eixo. Então, se você configurá-la, ela irá trocar as coordenadas, deixando a ordem sendo lat lon em vez do banco de dados.
- 32: Gera a caixa da geometria (envelope).

O argumento 'namespace prefix' pode ser usado para especificar um namespace prefix personalizado ou nenhum prefixo (se vazio). Se nulo ou omitido, o prefixo 'gml' é usado

Disponibilidade: 1.3.2

Disponibilidade: 1.5.0 suporte para geografia foi introduzido.

Melhorias: 2.0.0 prefixo suportado foi introduzido. A opção 4 para o GML3 foi introduzida para permitir a utilização da LineString em vez da tag Curva para linhas. O suporte GML3 para superfícies poliédricas e TINS foi introduzidos. A Opção 32 foi introduzida para gerar a caixa.

Alterações: 2.0.0 use argumentos nomeados por padrão

Melhorias: 2.1.0 suporte para id foi introduzido, para GML 3.

Note
Somente a versão 3+ de ST_AsGML suporta superfícies poliédricas e TINS.

 This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 17.2

 This function supports 3d and will not drop the z-index.

 This function supports Polyhedral surfaces.

 This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

Exemplos: Versão 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
>
```

Exemplos: Versão 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)) )'));
        st_asgml
```

```
        --------
 <gml:PolyhedralSurface>
<gml:polygonPatches>
   <gml:PolygonPatch>
                <gml:exterior>
                          <gml:LinearRing>
                                    <gml:posList srsDimension="3"
>0 0 0 0 0 1 0 1 1 0 1 0 0 0 0</gml:posList>
                          </gml:LinearRing>
                </gml:exterior>
   </gml:PolygonPatch>
   <gml:PolygonPatch>
                <gml:exterior>
                          <gml:LinearRing>
                                    <gml:posList srsDimension="3"
>0 0 0 0 1 0 1 1 0 1 0 0 0 0 0</gml:posList>
                          </gml:LinearRing>
                </gml:exterior>
   </gml:PolygonPatch>
   <gml:PolygonPatch>
                <gml:exterior>
                          <gml:LinearRing>
                                    <gml:posList srsDimension="3"
>0 0 0 1 0 0 1 0 1 0 0 1 0 0 0</gml:posList>
                          </gml:LinearRing>
                </gml:exterior>
   </gml:PolygonPatch>
   <gml:PolygonPatch>
                <gml:exterior>
                          <gml:LinearRing>
                                    <gml:posList srsDimension="3"
>1 1 0 1 1 1 1 0 1 1 0 0 1 1 0</gml:posList>
                          </gml:LinearRing>
                </gml:exterior>
   </gml:PolygonPatch>
   <gml:PolygonPatch>
                <gml:exterior>
                          <gml:LinearRing>
                                    <gml:posList srsDimension="3"
>0 1 0 0 1 1 1 1 1 1 1 0 0 1 0</gml:posList>
                          </gml:LinearRing>
                </gml:exterior>
   </gml:PolygonPatch>
   <gml:PolygonPatch>
                <gml:exterior>
                          <gml:LinearRing>
                                    <gml:posList srsDimension="3"
>0 0 1 1 0 1 1 1 1 0 1 1 0 0 1</gml:posList>
                          </gml:LinearRing>
                </gml:exterior>
   </gml:PolygonPatch>
</gml:polygonPatches>
</gml:PolyhedralSurface
>
```

Veja também.

ST_GeomFromGML

7.9.3.6 ST_AsKML

ST_AsKML — Retorna a geometria como uma versão GML com 2 ou 3 elementos.

Synopsis

text **ST_AsKML**(geometry geom, integer maxdecimaldigits=15, text nprefix=NULL);
text **ST_AsKML**(geography geog, integer maxdecimaldigits=15, text nprefix=NULL);

Descrição

Retorna a geometria como um elemento Keyhole Markup Language (KML). Existem muitas variantes desta função. Número máximo de casas decimais usado na saída (padrão 15), versão para 2 e o namespace não tem prefixo.

Warning
Using the `maxdecimaldigits` parameter can cause output geometry to become invalid. To avoid this use ST_ReducePrecision with a suitable gridsize first.

Note
Requer que PostGIS seja compilado com o suporte Porj. Use PostGIS_Full_Version para confirmar que você o suporte proj compilado.

Note
Disponibilidade: 1.2.2 - variantes futuras que incluem parâmetro versão que veio em 1.3.2

Note
Melhorias: 2.0.0 - Adiciona namespace prefixo. O padrão é não ter nenhum prefixo

Note
Changed: 3.0.0 - Removed the "versioned" variant signature

Note
A saída AsKML não funcionará com geometrias que não possuem um SRID

This function supports 3d and will not drop the z-index.

Examples

```
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>
```

Veja também.

ST_AsSVG, ST_AsGML

7.9.3.7 ST_AsLatLonText

ST_AsLatLonText — Retorna a representação de Graus, Minutos, Segundos do ponto dado.

Synopsis

text **ST_AsLatLonText**(geometry pt, text format='');

Descrição

Returns the Degrees, Minutes, Seconds representation of the point.

Note
É suposto que o ponto é uma projeção lat/lon. As coordenadas X (lon) e Y (lat), são normalizadas na saída para o alcance "normal" (-180 to +180 para lon, -90 para +90 para lat).

O texto parâmetro é um formato string que contém o formato do texto resultante, parecido com uma string de formato data. Tokens válidos são "D" para graus, "M" para minutos, "S" para segundos e "C" para direções cardiais (NSLO). Os tokens DMS podem se repetir para indicar a largura e precisão desejadas ("SSS.SSSS" significa " 1.0023").

"M", "S", e "C" são opcionais. Se "C" estiverem omitidas, os graus são mostrados com um "-" se sul ou oeste. Se "S" estiver omitido, os minutos serão mostrados como decimais com com tanta precisão de dígitos quanto você especificar. Se "M" também estiver omitido, os graus serão mostrados como decimais com tanta precisão de dígitos quanto você especificar.

Se a string formato for omitida (ou tiver tamanho zero) um formato padrão será usado.

Disponibilidade: 2.0

Examples

Formato padrão.

```
SELECT (ST_AsLatLonText('POINT (-3.2342342 -2.32498)'));
      st_aslatlontext
---------------------------
 2\textdegree{}19'29.928"S 3\textdegree{}14'3.243"W
```

Fornecendo um formato (o mesmo do padrão).

```
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
```

Outros caracteres além de D, M, S, C e . são somente passados.

```
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
```

Graus assinados em vez de direções cardiais.

```
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"
```

Graus decimais.

```
SELECT (ST_AsLatLonText('POINT (-3.2342342 -2.32498)', 'D.DDDD degrees C'));
          st_aslatlontext
----------------------------------
 2.3250 degrees S 3.2342 degrees W
```

Valores excessivamente grandes são normalizados.

```
SELECT (ST_AsLatLonText('POINT (-302.2342342 -792.32498)'));
        st_aslatlontext
-----------------------------
 72\textdegree{}19'29.928"S 57\textdegree{}45'56.757"E
```

7.9.3.8 ST_AsMARC21

ST_AsMARC21 — Returns geometry as a MARC21/XML record with a geographic datafield (034).

Synopsis

text **ST_AsMARC21** (geometry geom , text format='hdddmmss');

Descrição

This function returns a MARC21/XML record with Coded Cartographic Mathematical Data representing the bounding box of a given geometry. The `format` parameter allows to encode the coordinates in subfields `$d`,`$e`,`$f` and `$g` in all formats supported by the MARC21/XML standard. Valid formats are:

- cardinal direction, degrees, minutes and seconds (default): `hdddmmss`
- decimal degrees with cardinal direction: `hddd.dddddd`
- decimal degrees without cardinal direction: `ddd.dddddd`
- decimal minutes with cardinal direction: `hdddmm.mmmm`
- decimal minutes without cardinal direction: `dddmm.mmmm`
- decimal seconds with cardinal direction: `hdddmmss.sss`

The decimal sign may be also a comma, e.g. `hdddmm,mmmm`.

The precision of decimal formats can be limited by the number of characters after the decimal sign, e.g. `hdddmm.mm` for decimal minutes with a precision of two decimals.

This function ignores the Z and M dimensions.

LOC MARC21/XML versions supported:

- MARC21/XML 1.1

Availability: 3.3.0

Note
This function does not support non lon/lat geometries, as they are not supported by the MARC21/XML standard (Coded Cartographic Mathematical Data).

Note
The MARC21/XML Standard does not provide any means to annotate the spatial reference system for Coded Cartographic Mathematical Data, which means that this information will be lost after conversion to MARC21/XML.

Examples

Converting a `POINT` to MARC21/XML formated as hdddmmss (default)

```
                SELECT ST_AsMARC21('SRID=4326;POINT(-4.504289 54.253312)'::geometry);

                                st_asmarc21
                ------------------------------------------------
                <record xmlns="http://www.loc.gov/MARC21/slim">
                    <datafield tag="034" ind1="1" ind2=" ">
                        <subfield code="a"
>a</subfield>
                        <subfield code="d"
>W0043015</subfield>
                        <subfield code="e"
>W0043015</subfield>
                        <subfield code="f"
>N0541512</subfield>
                        <subfield code="g"
>N0541512</subfield>
                    </datafield>
                </record>
```

Converting a POLYGON to MARC21/XML formated in decimal degrees

```
                SELECT ST_AsMARC21('SRID=4326;POLYGON((-4.5792388916015625  ↩
                    54.18172660239091,-4.56756591796875  ↩
                    54.196993557130355,-4.546623229980469  ↩
                    54.18313300502024,-4.5792388916015625 54.18172660239091))'::geometry,' ↩
                    hddd.dddd');

                <record xmlns="http://www.loc.gov/MARC21/slim">
                    <datafield tag="034" ind1="1" ind2=" ">
                        <subfield code="a"
>a</subfield>
                        <subfield code="d"
>W004.5792</subfield>
                        <subfield code="e"
>W004.5466</subfield>
                        <subfield code="f"
>N054.1970</subfield>
                        <subfield code="g"
>N054.1817</subfield>
                    </datafield>
                </record>
```

Converting a GEOMETRYCOLLECTION to MARC21/XML formated in decimal minutes. The geometries order in the MARC21/XML output correspond to their order in the collection.

```
                SELECT ST_AsMARC21('SRID=4326;GEOMETRYCOLLECTION(POLYGON((13.1  ↩
                    52.65,13.516666666666667 52.65,13.516666666666667 52.38333333333333,13.1 ↩
                     52.38333333333333,13.1 52.65)),POINT(-4.5 54.25))'::geometry,'hdddmm. ↩
                    mmmm');

                               st_asmarc21
                -------------------------------------------------
                <record xmlns="http://www.loc.gov/MARC21/slim">
                    <datafield tag="034" ind1="1" ind2=" ">
                        <subfield code="a"
>a</subfield>
                        <subfield code="d"
>E01307.0000</subfield>
                        <subfield code="e"
>E01331.0000</subfield>
                        <subfield code="f"
>N05240.0000</subfield>
                        <subfield code="g"
>N05224.0000</subfield>
                    </datafield>
                    <datafield tag="034" ind1="1" ind2=" ">
                        <subfield code="a"
>a</subfield>
                        <subfield code="d"
>W00430.0000</subfield>
                        <subfield code="e"
>W00430.0000</subfield>
                        <subfield code="f"
>N05415.0000</subfield>
                        <subfield code="g"
>N05415.0000</subfield>
                    </datafield>
```

```
                </record>
```

Veja também.

ST_GeomFromMARC21

7.9.3.9 ST_AsMVTGeom

ST_AsMVTGeom — Transforms a geometry into the coordinate space of a MVT tile.

Synopsis

geometry **ST_AsMVTGeom**(geometry geom, box2d bounds, integer extent=4096, integer buffer=256, boolean clip_geom=true);

Descrição

Transforms a geometry into the coordinate space of a MVT (Mapbox Vector Tile) tile, clipping it to the tile bounds if required. The geometry must be in the coordinate system of the target map (using ST_Transform if needed). Commonly this is Web Mercator (SRID:3857).

The function attempts to preserve geometry validity, and corrects it if needed. This may cause the result geometry to collapse to a lower dimension.

The rectangular bounds of the tile in the target map coordinate space must be provided, so the geometry can be transformed, and clipped if required. The bounds can be generated using ST_MakeEnvelope.

This function is used to convert geometry into the tile coordinate space required by ST_AsMVT.

`geom` is the geometry to transform, in the coordinate system of the target map.

`bounds` is the rectangular bounds of the tile in map coordinate space, with no buffer.

`extent` is the tile extent size in tile coordinate space as defined by the MVT specification. Defaults to 4096.

`buffer` is the buffer size in tile coordinate space for geometry clippig. Defaults to 256.

`clip_geom` is a boolean to control if geometries are clipped or encoded as-is. Defaults to true.

Availability: 2.4.0

Note
From 3.0, Wagyu can be chosen at configure time to clip and validate MVT polygons. This library is faster and produces more correct results than the GEOS default, but it might drop small polygons.

Examples

```
SELECT ST_AsText(ST_AsMVTGeom(
        ST_GeomFromText('POLYGON ((0 0, 10 0, 10 5, 0 -5, 0 0))'),
        ST_MakeBox2D(ST_Point(0, 0), ST_Point(4096, 4096)),
        4096, 0, false));
                             st_astext
---------------------------------------------------------------------
 MULTIPOLYGON(((5 4096,10 4091,10 4096,5 4096)),((5 4096,0 4101,0 4096,5 4096)))
```

Canonical example for a Web Mercator tile using a computed tile bounds to query and clip geometry.

```
SELECT ST_AsMVTGeom(
            ST_Transform( geom, 3857 ),
            ST_TileEnvelope(12, 513, 412), extent =
> 4096, buffer =
> 64) AS geom
  FROM data
  WHERE geom && ST_TileEnvelope(12, 513, 412, margin =
> (64.0 / 4096))
```

Veja também.

ST_AsMVT, ST_MakeEnvelope, PostGIS_Wagyu_Version

7.9.3.10 ST_AsMVT

ST_AsMVT — Aggregate function returning a MVT representation of a set of rows.

Synopsis

bytea **ST_AsMVT**(anyelement set row);
bytea **ST_AsMVT**(anyelement row, text name);
bytea **ST_AsMVT**(anyelement row, text name, integer extent);
bytea **ST_AsMVT**(anyelement row, text name, integer extent, text geom_name);
bytea **ST_AsMVT**(anyelement row, text name, integer extent, text geom_name, text feature_id_name);

Descrição

An aggregate function which returns a binary Mapbox Vector Tile representation of a set of rows corresponding to a tile layer. The rows must contain a geometry column which will be encoded as a feature geometry. The geometry must be in tile coordinate space and valid as per the MVT specification. ST_AsMVTGeom can be used to transform geometry into tile coordinate space. Other row columns are encoded as feature attributes.

The Mapbox Vector Tile format can store features with varying sets of attributes. To use this capability supply a JSONB column in the row data containing Json objects one level deep. The keys and values in the JSONB values will be encoded as feature attributes.

Tiles with multiple layers can be created by concatenating multiple calls to this function using `||` or `STRING_AGG`.

Important
Do not call with a `GEOMETRYCOLLECTION` as an element in the row. However you can use ST_AsMVTGeom to prepare a geometry collection for inclusion.

`row` row data with at least a geometry column.

`name` is the name of the layer. Default is the string "default".

`extent` is the tile extent in screen space as defined by the specification. Default is 4096.

`geom_name` is the name of the geometry column in the row data. Default is the first geometry column. Note that PostgreSQL by default automatically folds unquoted identifiers to lower case, which means that unless the geometry column is quoted, e.g. `"MyMVTGeom"`, this parameter must be provided as lowercase.

`feature_id_name` is the name of the Feature ID column in the row data. If NULL or negative the Feature ID is not set. The first column matching name and valid type (smallint, integer, bigint) will be used as Feature ID, and any subsequent column will be added as a property. JSON properties are not supported.

Enhanced: 3.0 - added support for Feature ID.

Enhanced: 2.5.0 - added support parallel query.

Availability: 2.4.0

Examples

```
WITH mvtgeom AS
(
  SELECT ST_AsMVTGeom(geom, ST_TileEnvelope(12, 513, 412), extent =
> 4096, buffer =
> 64) AS geom, name, description
  FROM points_of_interest
  WHERE geom && ST_TileEnvelope(12, 513, 412, margin =
> (64.0 / 4096))
)
SELECT ST_AsMVT(mvtgeom.*)
FROM mvtgeom;
```

Veja também.

ST_AsMVTGeom, ST_MakeEnvelope

7.9.3.11 ST_AsSVG

ST_AsSVG — Returns SVG path data for a geometry.

Synopsis

text **ST_AsSVG**(geometry geom, integer rel=0, integer maxdecimaldigits=15);
text **ST_AsSVG**(geography geog, integer rel=0, integer maxdecimaldigits=15);

Descrição

Retorna a geometria como dados Scalar Vector Graphics (SVG). Use 1 como segundo argumento para ter os dados path implementados em termo de movimentos relacionados, o padrão (ou 0) utiliza movimento absolutos. O terceiro argumento pode ser usado para reduzir o máximo número de dígitos decimais usados na saída (padrão 15). Geometrias pontuais, serão renderizadas como cx/cy quando o argumento 'rel' for 0, x/y quando 'rel' for 1. Geometrias multipontuais são delimitadas por vírgulas (","). As geometrias GeometryCollection são delimitadas por ponto e vírgula (";").

For working with PostGIS SVG graphics, checkout pg_svg library which provides plpgsql functions for working with outputs from ST_AsSVG.

Enhanced: 3.4.0 to support all curve types

Alterações: 2.0.0 para usar args padrão e suporta args nomeados

Note
Disponibilidade: 1.2.2. Disponibilidade: 1.4.0 Alterado em PostGIS 1.4.0 para incluir comando L em path absoluto para entrar em conformidade com http://www.w3.org/TR/SVG/paths.html#PathDataBNF

 This method supports Circular Strings and Curves.

Examples

```
SELECT ST_AsSVG('POLYGON((0 0,0 1,1 1,1 0,0 0))'::geometry);

st_assvg
--------
M 0 0 L 0 -1 1 -1 1 0 Z
```

Circular string

```
SELECT ST_AsSVG( ST_GeomFromText('CIRCULARSTRING(-2 0,0 2,2 0,0 2,2 4)') );

st_assvg
--------
M -2 0 A 2 2 0 0 1 2 0 A 2 2 0 0 1 2 -4
```

Multi-curve

```
SELECT ST_AsSVG('MULTICURVE((5 5,3 5,3 3,0 3),
 CIRCULARSTRING(0 0,2 1,2 2))'::geometry, 0, 0);
 st_assvg
-----------------------------------------------
 M 5 -5 L 3 -5 3 -3 0 -3 M 0 0 A 2 2 0 0 0 2 -2
```

Multi-surface

```
SELECT ST_AsSVG('MULTISURFACE(
CURVEPOLYGON(CIRCULARSTRING(-2 0,-1 -1,0 0,1 -1,2 0,0 2,-2 0),
    (-1 0,0 0.5,1 0,0 1,-1 0)),
((7 8,10 10,6 14,4 11,7 8)))'::geometry, 0, 2);

st_assvg
-----------------------------------------------------------
M -2 0 A 1 1 0 0 0 0 0 A 1 1 0 0 0 2 0 A 2 2 0 0 0 -2 0 Z
M -1 0 L 0 -0.5 1 0 0 -1 -1 0 Z
M 7 -8 L 10 -10 6 -14 4 -11 Z
```

7.9.3.12 ST_AsTWKB

ST_AsTWKB — Retorna a geometria como TWKB, também conhecido como "Tiny Well-Known Binary"

Synopsis

bytea **ST_AsTWKB**(geometry geom, integer prec=0, integer prec_z=0, integer prec_m=0, boolean with_sizes=false, boolean with_boxes=false);
bytea **ST_AsTWKB**(geometry[] geom, bigint[] ids, integer prec=0, integer prec_z=0, integer prec_m=0, boolean with_sizes=false, boolean with_boxes=false);

Descrição

Retorna a geometria no formato TWKB (Tiny Well-Known Binary). TWKB é um compressed binary format com foco em minimizar o tamanho da saída.

Os parâmetros de dígitos decimais controlam quanta precisão está armazenada na saída. Por padrão, valores são arredondados para a unidade mais próxima antes de encoding. Por exemplo: um valor de 1 implica que o primeiro dígito a direita do ponto decimal será preservado.

Os tamanhos e os parâmetros das caixas limitadoras controlam onde as informações opcionais sobre o tamanho do encoding do objeto e os limites do objeto estão incluídas na saída. Por padrão elas não estão. Não as inclua a menos que o software do seu cliente tenha um uso para elas, como elas só ocupa espaço (e economizar espaço é o objeto do TWKB).

A forma arranjo entrada da função é usada para converter uma coleção de geometrias e identificadores únicos em uma coleção TWKB que preserva os identificadores. Isto é útil para clientes que esperam desempacotar uma coleção e acessar informações futuras sobre os objetos que estão dentro. Você pode criar os arranjos usando a função array_agg. Os outros parâmetros funcionam da mesma forma para o formato simples da função.

Note
O formato de especificação está disponível online em https://github.com/TWKB/Specification, e o código para construir um cliente JavaScript pode ser encontrado em https://github.com/TWKB/twkb.js.

Enhanced: 2.4.0 memory and speed improvements.

Disponibilidade: 2.2.0

Examples

```
SELECT ST_AsTWKB('LINESTRING(1 1,5 5)'::geometry);
                 st_astwkb
-------------------------------------------
\x02000202020808
```

Para criar um objeto TWKB agregado, incluir identificadores agrega as geometrias e objetos desejado primeiro, utilizando "array_agg()", então, utilize a função TWKB apropriada.

```
SELECT ST_AsTWKB(array_agg(geom), array_agg(gid)) FROM mytable;
                 st_astwkb
-------------------------------------------
\x040402020400000202
```

Veja também.

ST_GeomFromTWKB, ST_AsBinary, ST_AsEWKB, ST_AsEWKT, ST_GeomFromText

7.9.3.13 ST_AsX3D

ST_AsX3D — Retorna uma geometria em X3D nó xml formato do elemento: ISO-IEC-19776-1.2-X3DEncodings-XML

Synopsis

text **ST_AsX3D**(geometry g1, integer maxdecimaldigits=15, integer options=0);

Descrição

Retorna uma geometria como um elemento nó formatado X3D xml http://www.web3d.org/standards/number/19776-1. Se `maxdecimal` (precisão) não estiver especificada, então, leva para 15.

Note

Existem vários motivos para traduzir as geometrias PostGIS para X3D já que os tipos de geometria X3D não mapeiam diretamente para os tipos de geometria do PostGIS e alguns tipos X3D mais novos, que podem ser os melhores mapeadores que estávamos evitando já que a maioria das ferramentas renderizadoras não suportam eles. Sinta-se livre para postar um comentário se você tiver ideias de como podemos permitir as pessoas a indicarem seus mapeamentos preferidos.
Abaixo está como nós mapeamos os tipos 2D/3D do PostGIS para os tipos X3D, no momento

O argumento 'opções' é um bitfield. Para o PostGIS 2.2+, isto é usado para indicar onde representar as coordenadas atuais com o nó X3D GeoCoordinates Geospatial e, além disso, onde derrubar os eixos x/y. Por padrão, `ST_AsX3D` gera na forma de banco de dados (long,lat or X,Y), mas X3D de lat/lon, y/x podem ser preferidos.

- 0: X/Y na ordem de banco de dados (ex: ling/lat = X,Y é a ordem padrão de banco de dados), valor padrão e coordenadas não-espaciais (somente coordenada tag antiga).
- 1: Lançar X e Y. Se usado em conjunção com a opção de trocar a geocoordenada, então, a saída será "latitude_first" e as coordenadas serão lançadas também.
- 2: Gera coordenadas no GeoSpatial GeoCoordinates. Esta opção lançará um erro se as geometrias não estiverem na WGS 84 long lat (srid: 4326). Este é o único tipo GeoCoordinate suportado. Refer to X3D specs specifying a spatial reference system.. Saída padrão será: `GeoCoordinate geoSystem='"GD" "WE" "longitude_first"'`. If you prefer the X3D default of `GeoCoordinate geoSystem='"GD" "WE" "latitude_first"'` use `(2 + 1) = 3`

Tipo PostGIS	Tipo 2D X3D	Tipo 3D X3D
LINESTRING	ainda não foi implementado - será PoliLinha2D	LineSet
MULTILINESTRING	ainda não foi implementado - será PoliLinha2D	IndexedLineSet
MULTIPONTO	Poliponto2D	PointSet
PONTO	gera as coordenadas delimitadas pelo espaço	gera as coordenadas delimitadas pelo espaço
(MULTI) POLÍGONO, SUPERFÍCIE POLIÉDRICA	Marcação X3D inválida	IndexedFaceSet (anéis interiores atualmente gerados como outro faceset)
TIN	TriangleSet2D (ainda não implementado)	IndexedTriangleSet

Note

O suporte para geometrias 2D ainda não está completo. Os anéis interiores apenas desenhados como polígonos separados. Estamos trabalhando nisto.

Lots of advancements happening in 3D space particularly with X3D Integration with HTML5

Existe uma ótima fonte de visualizador X3D que você pode usar para ver as geometrias renderizadas. Free Wrl http://freewrl.sourceforge. binários para Mac, Linux, and Windows. Use FreeWRL_Launcher compactados para visualizar as geometrias.

Also check out PostGIS minimalist X3D viewer that utilizes this function and x3dDom html/js open source toolkit.

Disponibilidade: 2.0.0: ISO-IEC-19776-1.2-X3DEncodings-XML

Melhorias: 2.2.0: Suporte para GeoCoordinates e eixos (x/y, long/lat) lançando. Observe as opções para mais detalhes.

 This function supports 3d and will not drop the z-index.

 This function supports Polyhedral surfaces.

 This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

Exemplo: Cria um documento X3D completamente funcional - Isto irá gerar um cubo visível no FreeWrl e outros visualizadores X3D.

```
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
               --------
<?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
>
```

PostGIS buildings

Copy and paste the output of this query to x3d scene viewer and click Show

```
SELECT string_agg('<Shape
>' || ST_AsX3D(ST_Extrude(geom, 0,0, i*0.5)) ||
    '<Appearance>
          <Material diffuseColor="' || (0.01*i)::text || ' 0.8 0.2" specularColor="' ||  ↩
              (0.05*i)::text || ' 0 0.5"/>
```

```
        </Appearance>
    </Shape
>', '')
FROM ST_Subdivide(ST_Letters('PostGIS'),20) WITH ORDINALITY AS f(geom,i);
```

Buildings formed by subdividing PostGIS and extrusion

Exemplo: Um octógono elevado 3 unidades e com precisão decimal de 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
>
```

Exemplo: 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
>
```

Exemplo: Multilinestring fechada (o limite de um polígono com buracos)

```
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
>
```

7.9.3.14 ST_GeoHash

ST_GeoHash — Retorna uma representação GeoHash da geometria.

Synopsis

text **ST_GeoHash**(geometry geom, integer maxchars=full_precision_of_point);

Descrição

Computes a GeoHash representation of a geometry. A GeoHash encodes a geographic Point into a text form that is sortable and searchable based on prefixing. A shorter GeoHash is a less precise representation of a point. It can be thought of as a box that contains the point.

Non-point geometry values with non-zero extent can also be mapped to GeoHash codes. The precision of the code depends on the geographic extent of the geometry.

If `maxchars` is not specified, the returned GeoHash code is for the smallest cell containing the input geometry. Points return a GeoHash with 20 characters of precision (about enough to hold the full double precision of the input). Other geometric types may return a GeoHash with less precision, depending on the extent of the geometry. Larger geometries are represented with less precision, smaller ones with more precision. The box determined by the GeoHash code always contains the input feature.

If `maxchars` is specified the returned GeoHash code has at most that many characters. It maps to a (possibly) lower precision representation of the input geometry. For non-points, the starting point of the calculation is the center of the bounding box of the geometry.

Disponibilidade: 1.4.0

Note
ST_GeoHash requires input geometry to be in geographic (lon/lat) coordinates.

 This method supports Circular Strings and Curves.

Examples

```
SELECT ST_GeoHash( ST_Point(-126,48) );

         st_geohash
----------------------
 c0w3hf1s70w3hf1s70w3

SELECT ST_GeoHash( ST_Point(-126,48), 5);

 st_geohash
------------
 c0w3h

-- This line contains the point, so the GeoHash is a prefix of the point code
SELECT ST_GeoHash('LINESTRING(-126 48, -126.1 48.1)'::geometry);

 st_geohash
------------
 c0w3
```

Veja também.

ST_GeomFromGeoHash, ST_PointFromGeoHash, ST_Box2dFromGeoHash

7.10 Operadores

7.10.1 Bounding Box Operators

7.10.1.1 &&

&& — Retorna VERDADE se a caixa limitadora 2D de A intersecta a caixa limitadora 2D de B.

Synopsis

boolean **&&**(geometry A , geometry B);
boolean **&&**(geography A , geography B);

Descrição

O operador && retorna VERDADE se a caixa limitadora 2D da geometria A intersecta a caixa limitadora 2D da geometria B.

Note
Esse operador fará uso de qualquer um dos indexes que talvez estejam disponíveis nas geometrias.

Melhorias: 2.0.0 suporte a superfícies poliédricas foi introduzido.

Disponibilidade: 1.5.0 Suporte para geografia foi introduzido

 This method supports Circular Strings and Curves.

 This function supports Polyhedral surfaces.

Exemplos

```
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)
```

Veja também.

ST_Intersects, &>, &<|, &<, ~, @

7.10.1.2 &&(geometry,box2df)

&&(geometry,box2df) — Returns TRUE if a geometry's (cached) 2D bounding box intersects a 2D float precision bounding box (BOX2DF).

Synopsis

boolean **&&**(geometry A , box2df B);

Descrição

The && operator returns TRUE if the cached 2D bounding box of geometry A intersects the 2D bounding box B, using float precision. This means that if B is a (double precision) box2d, it will be internally converted to a float precision 2D bounding box (BOX2DF)

Note
This operand is intended to be used internally by BRIN indexes, more than by users.

Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+.

 This method supports Circular Strings and Curves.

 This function supports Polyhedral surfaces.

Exemplos

```
SELECT ST_Point(1,1) && ST_MakeBox2D(ST_Point(0,0), ST_Point(2,2)) AS overlaps;

 overlaps
----------
 t
(1 row)
```

Veja também.

&&(box2df,geometry), &&(box2df,box2df), ~(geometry,box2df), ~(box2df,geometry), ~(box2df,box2df), @(geometry,box2df), @(box2df,geometry), @(box2df,box2df)

7.10.1.3 &&(box2df,geometry)

&&(box2df,geometry) — Returns `TRUE` if a 2D float precision bounding box (BOX2DF) intersects a geometry's (cached) 2D bounding box.

Synopsis

boolean **&&**(box2df A , geometry B);

Descrição

The `&&` operator returns `TRUE` if the 2D bounding box A intersects the cached 2D bounding box of geometry B, using float precision. This means that if A is a (double precision) box2d, it will be internally converted to a float precision 2D bounding box (BOX2DF)

Note
This operand is intended to be used internally by BRIN indexes, more than by users.

Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+.


This method supports Circular Strings and Curves.


This function supports Polyhedral surfaces.

Exemplos

```
SELECT ST_MakeBox2D(ST_Point(0,0), ST_Point(2,2)) && ST_Point(1,1) AS overlaps;

 overlaps
----------
 t
(1 row)
```

Veja também.

&&(geometry,box2df), &&(box2df,box2df), ~(geometry,box2df), ~(box2df,geometry), ~(box2df,box2df), @(geometry,box2df), @(box2df,geometry), @(box2df,box2df)

7.10.1.4 &&(box2df,box2df)

&&(box2df,box2df) — Returns `TRUE` if two 2D float precision bounding boxes (BOX2DF) intersect each other.

Synopsis

boolean **&&**(box2df A , box2df B);

Descrição

The `&&` operator returns `TRUE` if two 2D bounding boxes A and B intersect each other, using float precision. This means that if A (or B) is a (double precision) box2d, it will be internally converted to a float precision 2D bounding box (BOX2DF)

Note
This operator is intended to be used internally by BRIN indexes, more than by users.

Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+.

 This method supports Circular Strings and Curves.

 This function supports Polyhedral surfaces.

Exemplos

```
SELECT ST_MakeBox2D(ST_Point(0,0), ST_Point(2,2)) && ST_MakeBox2D(ST_Point(1,1), ST_Point ↩
    (3,3)) AS overlaps;

 overlaps
----------
 t
(1 row)
```

Veja também.

&&(geometry,box2df), &&(box2df,geometry), ~(geometry,box2df), ~(box2df,geometry), ~(box2df,box2df), @(geometry,box2df), @(box2df,geometry), @(box2df,box2df)

7.10.1.5 &&&

&&& — Retorna `VERDADE` se a caixa limitadora n-D de A intersecta a caixa limitadora n-D de B.

Synopsis

boolean **&&&**(geometry A , geometry B);

Descrição

O operador `&&&` retorna `VERDADE` se a caixa limitadora n-D da geometria A intersecta a caixa limitadora n-D da geometria B.

Note
Esse operador fará uso de qualquer um dos indexes que talvez estejam disponíveis nas geometrias.

Disponibilidade: 2.0.0

 This method supports Circular Strings and Curves.

This function supports Polyhedral surfaces.

This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

This function supports 3d and will not drop the z-index.

Exemplos: LineStrings 3D

```
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           | t
       2 |       3 | f           | t
```

Exemplos: LineStrings 3M

```
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           | t
       2 |       3 | f           | t
```

Veja também.

&&

7.10.1.6 &&&(geometry,gidx)

&&&(geometry,gidx) — Returns `TRUE` if a geometry's (cached) n-D bounding box intersects a n-D float precision bounding box (GIDX).

Synopsis

boolean **&&&**(geometry A , gidx B);

Descrição

The `&&&` operator returns `TRUE` if the cached n-D bounding box of geometry A intersects the n-D bounding box B, using float precision. This means that if B is a (double precision) box3d, it will be internally converted to a float precision 3D bounding box (GIDX)

Note
This operator is intended to be used internally by BRIN indexes, more than by users.

Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+.

 This method supports Circular Strings and Curves.

 This function supports Polyhedral surfaces.

 This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

 This function supports 3d and will not drop the z-index.

Exemplos

```
SELECT ST_MakePoint(1,1,1) &&& ST_3DMakeBox(ST_MakePoint(0,0,0), ST_MakePoint(2,2,2)) AS  ↩
    overlaps;

 overlaps
----------
 t
(1 row)
```

Veja também.

&&&(gidx,geometry), &&&(gidx,gidx)

7.10.1.7 &&&(gidx,geometry)

&&&(gidx,geometry) — Returns `TRUE` if a n-D float precision bounding box (GIDX) intersects a geometry's (cached) n-D bounding box.

Synopsis

boolean **&&&**(gidx A , geometry B);

Descrição

The `&&&` operator returns `TRUE` if the n-D bounding box A intersects the cached n-D bounding box of geometry B, using float precision. This means that if A is a (double precision) box3d, it will be internally converted to a float precision 3D bounding box (GIDX)

Note
This operator is intended to be used internally by BRIN indexes, more than by users.

Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+.

This method supports Circular Strings and Curves.

This function supports Polyhedral surfaces.

This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

This function supports 3d and will not drop the z-index.

Exemplos

```
SELECT ST_3DMakeBox(ST_MakePoint(0,0,0), ST_MakePoint(2,2,2)) &&& ST_MakePoint(1,1,1) AS  ↩
    overlaps;

 overlaps
----------
 t
(1 row)
```

Veja também.

&&&(geometry,gidx), &&&(gidx,gidx)

7.10.1.8 &&&(gidx,gidx)

&&&(gidx,gidx) — Returns `TRUE` if two n-D float precision bounding boxes (GIDX) intersect each other.

Synopsis

boolean **&&&**(gidx A , gidx B);

Descrição

The `&&&` operator returns `TRUE` if two n-D bounding boxes A and B intersect each other, using float precision. This means that if A (or B) is a (double precision) box3d, it will be internally converted to a float precision 3D bounding box (GIDX)

Note
This operator is intended to be used internally by BRIN indexes, more than by users.

Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+.

This method supports Circular Strings and Curves.

This function supports Polyhedral surfaces.

This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

This function supports 3d and will not drop the z-index.

Exemplos

```
SELECT ST_3DMakeBox(ST_MakePoint(0,0,0), ST_MakePoint(2,2,2)) &&& ST_3DMakeBox(ST_MakePoint ↩
    (1,1,1), ST_MakePoint(3,3,3)) AS overlaps;

 overlaps
----------
 t
(1 row)
```

Veja também.

&&&(geometry,gidx), &&&(gidx,geometry)

7.10.1.9 &<

&< — Retorna VERDADE se a caixa limitadora de A sobrepõe ou está à esquerda de B.

Synopsis

boolean **&<**(geometry A , geometry B);

Descrição

O operador &< retorna VERDADE se a caixa limitadora da geometria A sobrepõe ou está à esquerda da caixa da geometria B, ou mais precisamente, sobrepõe ou NÃO está à direita da caixa limitadora da geometria B.

Note
Esse operador fará uso de qualquer um dos indexes que talvez estejam disponíveis nas geometrias.

Exemplos

```
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)
```

Veja também.

&&, |&>, &>, &<|

7.10.1.10 &<|

&<| — Retorna VERDADE se a caixa limitadora de A sobrepõe ou está abaixo de B.

Synopsis

boolean **&<|**(geometry A , geometry B);

Descrição

O operador &<| retorna VERDADE se a caixa limitadora da geometria A sobrepõe ou está abaixo da caixa da geometria B, ou mais precisamente, sobrepõe ou NÃO está acima da caixa limitadora da geometria B.

 This method supports Circular Strings and Curves.

 This function supports Polyhedral surfaces.

Note
Esse operador fará uso de qualquer um dos indexes que talvez estejam disponíveis nas geometrias.

Exemplos

```
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)
```

Veja também.

&&, |&>, &>, &<

7.10.1.11 &>

&> — Retorna VERDADE se a caixa limitadora de A sobrepõe ou está à direita de B.

Synopsis

boolean **&>**(geometry A , geometry B);

Descrição

O operador &> retorna VERDADE se a caixa limitadora da geometria A sobrepõe ou está à direita da caixa da geometria B, ou mais precisamente, sobrepõe ou NÃO está à esquerda da caixa limitadora da geometria B.

Note
Esse operador fará uso de qualquer um dos indexes que talvez estejam disponíveis nas geometrias.

Exemplos

```
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)
```

Veja também.

&&, |&>, &<|, &<

7.10.1.12 <<

<< — Retorna VERDADE se uma caixa limitadora de A está estritamente à esquerda da de B.

Synopsis

boolean **<<**(geometry A , geometry B);

Descrição

O operador << retorna VERDADE se a caixa limitadora da geometria A está estritamente à esquerda da caixa limitadora da geometria B.

Note
Esse operador fará uso de qualquer um dos indexes que talvez estejam disponíveis nas geometrias.

Exemplos

```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 << tbl2.column2 AS left
FROM
  ( VALUES
        (1, 'LINESTRING (1 2, 1 5)'::geometry)) AS tbl1,
  ( VALUES
        (2, 'LINESTRING (0 0, 4 3)'::geometry),
        (3, 'LINESTRING (6 0, 6 5)'::geometry),
        (4, 'LINESTRING (2 2, 5 6)'::geometry)) AS tbl2;

 column1 | column1 | left
---------+---------+------
           1 |       2 | f
           1 |       3 | t
           1 |       4 | t
(3 rows)
```

Veja também.

>>, |>>, <<|

7.10.1.13 <<|

<<| — Retorna VERDADE se uma caixa limitadora de A está estritamente abaixo da de B.

Synopsis

boolean **<<|**(geometry A , geometry B);

Descrição

O operador <<| retorna VERDADE se a caixa limitadora da geometria A está estritamente à esquerda da caixa limitadora da geometria B.

Note

Esse operador fará uso de qualquer um dos indexes que talvez estejam disponíveis nas geometrias.

Exemplos

```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 <<| tbl2.column2 AS below
FROM
  ( VALUES
        (1, 'LINESTRING (0 0, 4 3)'::geometry)) AS tbl1,
  ( VALUES
        (2, 'LINESTRING (1 4, 1 7)'::geometry),
        (3, 'LINESTRING (6 1, 6 5)'::geometry),
        (4, 'LINESTRING (2 3, 5 6)'::geometry)) AS tbl2;

 column1 | column1 | below
---------+---------+-------
           1 |       2 | t
           1 |       3 | f
           1 |       4 | f
(3 rows)
```

Veja também.

<<, >>, |>>

7.10.1.14 =

= — Returns TRUE if the coordinates and coordinate order geometry/geography A are the same as the coordinates and coordinate order of geometry/geography B.

Synopsis

boolean =(geometry A , geometry B);
boolean =(geography A , geography B);

Descrição

The = operator returns TRUE if the coordinates and coordinate order geometry/geography A are the same as the coordinates and coordinate order of geometry/geography B. PostgreSQL uses the =, <, and > operators defined for geometries to perform internal orderings and comparison of geometries (ie. in a GROUP BY or ORDER BY clause).

Note
Only geometry/geography that are exactly equal in all respects, with the same coordinates, in the same order, are considered equal by this operator. For "spatial equality", that ignores things like coordinate order, and can detect features that cover the same spatial area with different representations, use ST_OrderingEquals or ST_Equals

Caution
This operand will NOT make use of any indexes that may be available on the geometries. For an index assisted exact equality test, combine = with &&.

Changed: 2.4.0, in prior versions this was bounding box equality not a geometric equality. If you need bounding box equality, use ~= instead.

 This method supports Circular Strings and Curves.

 This function supports Polyhedral surfaces.

Exemplos

```
SELECT 'LINESTRING(0 0, 0 1, 1 0)'::geometry = 'LINESTRING(1 1, 0 0)'::geometry;
 ?column?
----------
 f
(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)

-- Note: 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)
 LINESTRING(1 1,0 0)
(2 rows)

-- In versions prior to 2.0, this used to return true --
 SELECT ST_GeomFromText('POINT(1707296.37 4820536.77)') =
        ST_GeomFromText('POINT(1707296.27 4820536.87)') As pt_intersect;

--pt_intersect --
f
```

Veja também.

ST_Equals, ST_OrderingEquals, ~=

7.10.1.15 >>

>> — Returns TRUE if A's bounding box is strictly to the right of B's.

Synopsis

boolean **>>**(geometry A , geometry B);

Descrição

O operador >> retorna VERDADE se a caixa limitadora da geometria A está estritamente à direita da caixa limitadora da geometria B.

Note
Esse operador fará uso de qualquer um dos indexes que talvez estejam disponíveis nas geometrias.

Exemplos

```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 >> tbl2.column2 AS right
FROM
  ( VALUES
        (1, 'LINESTRING (2 3, 5 6)'::geometry)) AS tbl1,
  ( VALUES
        (2, 'LINESTRING (1 4, 1 7)'::geometry),
        (3, 'LINESTRING (6 1, 6 5)'::geometry),
        (4, 'LINESTRING (0 0, 4 3)'::geometry)) AS tbl2;

 column1 | column1 | right
```

```
---------+---------+-------
           1 |       2 | t
           1 |       3 | f
           1 |       4 | f
(3 rows)
```

Veja também.

<<, |>>, <<|

7.10.1.16 @

@ — Retorna VERDADE se uma caixa limitadora de A está contida pela de B.

Synopsis

boolean @(geometry A , geometry B);

Descrição

O operador @ retorna VERDADE se a caixa limitadora da geometria A estiver completamente contida pela caixa limitadora da geometria B.

Note
Esse operador fará uso de qualquer um dos indexes que talvez estejam disponíveis nas geometrias.

Exemplos

```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 @ tbl2.column2 AS contained
FROM
  ( VALUES
        (1, 'LINESTRING (1 1, 3 3)'::geometry)) AS tbl1,
  ( VALUES
        (2, 'LINESTRING (0 0, 4 4)'::geometry),
        (3, 'LINESTRING (2 2, 4 4)'::geometry),
        (4, 'LINESTRING (1 1, 3 3)'::geometry)) AS tbl2;

 column1 | column1 | contained
---------+---------+-----------
           1 |       2 | t
           1 |       3 | f
           1 |       4 | t
(3 rows)
```

Veja também.

~, &&

7.10.1.17 @(geometry,box2df)

@(geometry,box2df) — Returns TRUE if a geometry's 2D bounding box is contained into a 2D float precision bounding box (BOX2DF).

Synopsis

boolean @(geometry A , box2df B);

Descrição

The `@` operator returns `TRUE` if the A geometry's 2D bounding box is contained the 2D bounding box B, using float precision. This means that if B is a (double precision) box2d, it will be internally converted to a float precision 2D bounding box (BOX2DF)

Note
This operand is intended to be used internally by BRIN indexes, more than by users.

Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+.

 This method supports Circular Strings and Curves.

 This function supports Polyhedral surfaces.

Exemplos

```
SELECT ST_Buffer(ST_GeomFromText('POINT(2 2)'), 1) @ ST_MakeBox2D(ST_Point(0,0), ST_Point ↩
    (5,5)) AS is_contained;

 is_contained
--------------
 t
(1 row)
```

Veja também.

&&(geometry,box2df), &&(box2df,geometry), &&(box2df,box2df), ~(geometry,box2df), ~(box2df,geometry), ~(box2df,box2df), @(box2df,geometry), @(box2df,box2df)

7.10.1.18 @(box2df,geometry)

@(box2df,geometry) — Returns `TRUE` if a 2D float precision bounding box (BOX2DF) is contained into a geometry's 2D bounding box.

Synopsis

boolean @(box2df A , geometry B);

Descrição

The `@` operator returns `TRUE` if the 2D bounding box A is contained into the B geometry's 2D bounding box, using float precision. This means that if B is a (double precision) box2d, it will be internally converted to a float precision 2D bounding box (BOX2DF)

Note
This operand is intended to be used internally by BRIN indexes, more than by users.

Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+.

This method supports Circular Strings and Curves.

This function supports Polyhedral surfaces.

Exemplos

```
SELECT ST_MakeBox2D(ST_Point(2,2), ST_Point(3,3)) @ ST_Buffer(ST_GeomFromText('POINT(1 1)') ↩
    , 10) AS is_contained;

 is_contained
--------------
 t
(1 row)
```

Veja também.

&&(geometry,box2df), &&(box2df,geometry), &&(box2df,box2df), ~(geometry,box2df), ~(box2df,geometry), ~(box2df,box2df), @(geometry,box2df), @(box2df,box2df)

7.10.1.19 @(box2df,box2df)

@(box2df,box2df) — Returns `TRUE` if a 2D float precision bounding box (BOX2DF) is contained into another 2D float precision bounding box.

Synopsis

boolean @(box2df A , box2df B);

Descrição

The `@` operator returns `TRUE` if the 2D bounding box A is contained into the 2D bounding box B, using float precision. This means that if A (or B) is a (double precision) box2d, it will be internally converted to a float precision 2D bounding box (BOX2DF)

Note
This operand is intended to be used internally by BRIN indexes, more than by users.

Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+.

This method supports Circular Strings and Curves.

This function supports Polyhedral surfaces.

Exemplos

```
SELECT ST_MakeBox2D(ST_Point(2,2), ST_Point(3,3)) @ ST_MakeBox2D(ST_Point(0,0), ST_Point ↩
    (5,5)) AS is_contained;

 is_contained
--------------
 t
(1 row)
```

Veja também.

&&(geometry,box2df), &&(box2df,geometry), &&(box2df,box2df), ~(geometry,box2df), ~(box2df,geometry), ~(box2df,box2df), @(geometry,box2df), @(box2df,geometry)

7.10.1.20 |&>

|&> — Retorna VERDADE se a caixa limitadora de A sobrepõe ou está acima de B.

Synopsis

boolean **|&>**(geometry A , geometry B);

Descrição

O operador |&> retorna VERDADE se a caixa limitadora da geometria A sobrepõe ou está acima da caixa da geometria B, ou mais precisamente, sobrepõe ou NÃO está abaixo da caixa limitadora da geometria B.

Note
Esse operador fará uso de qualquer um dos indexes que talvez estejam disponíveis nas geometrias.

Exemplos

```
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)
```

Veja também.

&&, &>, &<|, &<

7.10.1.21 |>>

|>> — Retorna VERDADE se uma caixa limitadora de A está estritamente acima da de B.

Synopsis

boolean **|>>**(geometry A , geometry B);

Descrição

The |>> operator returns TRUE if the bounding box of geometry A is strictly above the bounding box of geometry B.

Note
Esse operador fará uso de qualquer um dos indexes que talvez estejam disponíveis nas geometrias.

Exemplos

```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 |>> tbl2.column2 AS above
FROM
  ( VALUES
        (1, 'LINESTRING (1 4, 1 7)'::geometry)) AS tbl1,
  ( VALUES
        (2, 'LINESTRING (0 0, 4 2)'::geometry),
        (3, 'LINESTRING (6 1, 6 5)'::geometry),
        (4, 'LINESTRING (2 3, 5 6)'::geometry)) AS tbl2;

 column1 | column1 | above
---------+---------+-------
           1 |       2 | t
           1 |       3 | f
           1 |       4 | f
(3 rows)
```

Veja também.

<<, >>, <<|

7.10.1.22 ~

~ — Retorna VERDADE se uma caixa limitadora de A contém a de B.

Synopsis

boolean ~(geometry A , geometry B);

Descrição

O operador ~ retorna VERDADE se a caixa limitadora da geometria A estiver completamente contida pela caixa limitadora da geometria B.

Note
Esse operador fará uso de qualquer um dos indexes que talvez estejam disponíveis nas geometrias.

Exemplos

```
SELECT tbl1.column1, tbl2.column1, tbl1.column2 ~ tbl2.column2 AS contains
FROM
  ( VALUES
        (1, 'LINESTRING (0 0, 3 3)'::geometry)) AS tbl1,
  ( VALUES
        (2, 'LINESTRING (0 0, 4 4)'::geometry),
        (3, 'LINESTRING (1 1, 2 2)'::geometry),
        (4, 'LINESTRING (0 0, 3 3)'::geometry)) AS tbl2;

 column1 | column1 | contains
---------+---------+----------
           1 |       2 | f
           1 |       3 | t
           1 |       4 | t
(3 rows)
```

Veja também.

@, &&

7.10.1.23 ~(geometry,box2df)

~(geometry,box2df) — Returns TRUE if a geometry's 2D bonding box contains a 2D float precision bounding box (GIDX).

Synopsis

boolean ~(geometry A , box2df B);

Descrição

The ~ operator returns TRUE if the 2D bounding box of a geometry A contains the 2D bounding box B, using float precision. This means that if B is a (double precision) box2d, it will be internally converted to a float precision 2D bounding box (BOX2DF)

Note
This operand is intended to be used internally by BRIN indexes, more than by users.

Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+.

 This method supports Circular Strings and Curves.

 This function supports Polyhedral surfaces.

Exemplos

```
SELECT ST_Buffer(ST_GeomFromText('POINT(1 1)'), 10) ~ ST_MakeBox2D(ST_Point(0,0), ST_Point ↩
    (2,2)) AS contains;

 contains
----------
 t
(1 row)
```

Veja também.

&&(geometry,box2df), &&(box2df,geometry), &&(box2df,box2df), ~(box2df,geometry), ~(box2df,box2df), @(geometry,box2df), @(box2df,geometry), @(box2df,box2df)

7.10.1.24 ~(box2df,geometry)

~(box2df,geometry) — Returns TRUE if a 2D float precision bounding box (BOX2DF) contains a geometry's 2D bonding box.

Synopsis

boolean ~(box2df A , geometry B);

Descrição

The ~ operator returns TRUE if the 2D bounding box A contains the B geometry's bounding box, using float precision. This means that if A is a (double precision) box2d, it will be internally converted to a float precision 2D bounding box (BOX2DF)

Note
This operand is intended to be used internally by BRIN indexes, more than by users.

Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+.

 This method supports Circular Strings and Curves.

 This function supports Polyhedral surfaces.

Exemplos

```
SELECT ST_MakeBox2D(ST_Point(0,0), ST_Point(5,5)) ~ ST_Buffer(ST_GeomFromText('POINT(2 2)') ↩
    , 1) AS contains;

 contains
----------
 t
(1 row)
```

Veja também.

&&(geometry,box2df), &&(box2df,geometry), &&(box2df,box2df), ~(geometry,box2df), ~(box2df,box2df), @(geometry,box2df), @(box2df,geometry), @(box2df,box2df)

7.10.1.25 ~(box2df,box2df)

~(box2df,box2df) — Returns TRUE if a 2D float precision bounding box (BOX2DF) contains another 2D float precision bounding box (BOX2DF).

Synopsis

boolean ~(box2df A , box2df B);

Descrição

The ~ operator returns TRUE if the 2D bounding box A contains the 2D bounding box B, using float precision. This means that if A is a (double precision) box2d, it will be internally converted to a float precision 2D bounding box (BOX2DF)

Note
This operand is intended to be used internally by BRIN indexes, more than by users.

Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+.

This method supports Circular Strings and Curves.

This function supports Polyhedral surfaces.

Exemplos

```
SELECT ST_MakeBox2D(ST_Point(0,0), ST_Point(5,5)) ~ ST_MakeBox2D(ST_Point(2,2), ST_Point ↩
    (3,3)) AS contains;

 contains
----------
 t
(1 row)
```

Veja também.

&&(geometry,box2df), &&(box2df,geometry), &&(box2df,box2df), ~(geometry,box2df), ~(box2df,geometry), @(geometry,box2df), @(box2df,geometry), @(box2df,box2df)

7.10.1.26 ~=

~= — Retorna VERDADE se a caixa limitadora de A é a mesma de B.

Synopsis

boolean **~=**(geometry A , geometry B);

Descrição

O operador ~ retorna VERDADE se a caixa limitadora da geometria/geografia A for a mesma da caixa limitadora da geometria/-geografia B.

Note
Esse operador fará uso de qualquer um dos indexes que talvez estejam disponíveis nas geometrias.

Disponibilidade: 1.5.0 comportamento alterado

This function supports Polyhedral surfaces.

Warning
This operator has changed behavior in PostGIS 1.5 from testing for actual geometric equality to only checking for bounding box equality. To complicate things it also depends on if you have done a hard or soft upgrade which behavior your database has. To find out which behavior your database has you can run the query below. To check for true equality use ST_OrderingEquals or ST_Equals.

Exemplos

```
select 'LINESTRING(0 0, 1 1)'::geometry ~= 'LINESTRING(0 1, 1 0)'::geometry as equality;
 equality   |
------------------+
          t     |
```

Veja também.

ST_Equals, ST_OrderingEquals, =

7.10.2 Operadores

7.10.2.1 <->

<-> — Retorna a distância 2D entre A e B.

Synopsis

double precision **<->**(geometry A , geometry B);
double precision **<->**(geography A , geography B);

Descrição

O operador <-> retorna a distância 2D entre duas geometrias. Usado nas orações "ORDEM" que fornecem configurações de resultado index-assisted nearest-neighbor. Para o PostgreSQL menor que 9.5 somente fornece a distância centroide das caixas limitadoras e para PostgreSQL 9.5+, a verdadeira distância KNN procura dando verdadeiras distâncias entre geometrias, e distância esférica para geografias.

Note
Esse operador fará uso dos indexes 2D GiST que podem estar disponíveis nas geometrias. É diferente de outros operadores que usam indexes espaciais em que eles só são usados quando o operador está na oração ORDEM.

Note
O index só rejeita se uma das geometrias é uma constante (não em uma subquery/cte). ex. 'SRID=3005;POINT(1011102 450541)'::geometria ao invés de uma .geom

Vá para OpenGeo workshop: Nearest-Neighbour Searching para um exemplo real.

melhorias: 2.2.0 -- Verdadeiro comportamento KNN ("vizinho mais perto de K") para geometria e geografia para PostgreSQL 9.5+. Note que para geografia o KNN é baseado em esfera ao invés de esferoide. Para o PostgreSQL 9.4 ou menor, o suporte para geografia é novo, mas só suporta caixa centroide.

Alterações: 2.2.0 -- Para usuários do PostgreSQL 9.5, a sintaxe Hybrid antiga pode ser ais lenta, então, você vai querer se livrar daquele hack se você está executando seu código só no PostGIS 2.2+ 9.5+. Veja os exemplos abaixo.

Disponibilidade: 2.0.0 -- O KNN mais fraco fornece vizinho mais próximos baseados em distâncias centroides de geometrias, ao invés de distâncias reais. Resultados corretos para pontos, incorretos para todos os outros tipos. Disponível para PostgreSQL 9.1+

Exemplos

```
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    | 128
 5541.57712511724 | ALQ    | 129A
 5579.67450712005 | ALQ    | 001
  6083.4207708641 | ALQ    | 131
  7691.2205404848 | ALQ    | 003
 7900.75451037313 | ALQ    | 122
 8694.20710669982 | ALQ    | 129B
 9564.24289057111 | ALQ    | 130
  12089.665931705 | ALQ    | 127
 18472.5531479404 | ALQ    | 002
(10 rows)
```

Então, a resposta KNN crua:

```
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    | 128
 5541.57712511724 | ALQ    | 129A
 5579.67450712005 | ALQ    | 001
  6083.4207708641 | ALQ    | 131
  7691.2205404848 | ALQ    | 003
 7900.75451037313 | ALQ    | 122
 8694.20710669982 | ALQ    | 129B
 9564.24289057111 | ALQ    | 130
  12089.665931705 | ALQ    | 127
 18472.5531479404 | ALQ    | 002
(10 rows)
```

Se você executar "ANÁLISE EXPLICATIVA" nas duas pesquisas, você verá uma apresentação melhorada para a segunda.

Para usuários com PostgreSQL < 9.5, use uma pesquisa hybrid para encontrar os vizinhos verdadeiros mais próximos. Primeiro, uma pesquisa CTE usando o index-assisted KNN, e depois, uma pesquisa exata para pegar a ordem certa:

```
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    | 128
 5541.57712511724 | ALQ    | 129A
 5579.67450712005 | ALQ    | 001
  6083.4207708641 | ALQ    | 131
  7691.2205404848 | ALQ    | 003
 7900.75451037313 | ALQ    | 122
 8694.20710669982 | ALQ    | 129B
 9564.24289057111 | ALQ    | 130
  12089.665931705 | ALQ    | 127
 18472.5531479404 | ALQ    | 002
(10 rows)
```

Veja também.

ST_DWithin, ST_Distance, <#>

7.10.2.2 |=|

|=| — Retorna a distância entre As trajetórias A e B ao ponto de aproximação mais perto.

Synopsis

double precision |=|(geometry A , geometry B);

Descrição

O operador |=| retorna a distância 3D entre duas trajetórias (Veja ST_IsValidTrajectory). Isso é o mesmo que ST_DistanceCPA, mas como um operador pode ser usado para fazer pesquisas de vizinhos próximos usando um index n-dimensional (requer PostgreSQL 9.5.0 ou superior).

Note
Esse operador fará uso dos indexes ND GiST que podem estar disponíveis nas geometrias. É diferente de outros operadores que usam indexes espaciais em que eles só são usados quando o operador está na oração ORDEM.

Note
O index só rejeita se uma das geometrias é uma constante (não em uma subquery/cte). ex.'SRID=3005;LINESTRINGM(0 0 0,0 0 1)'::geometria ao invés de uma .geom

Disponibilidade: 2.2.0. Index suportado disponível somente para PostgreSQL 9.5+

Exemplos

```
-- Save a literal query trajectory in a psql variable...
\set qt 'ST_AddMeasure(ST_MakeLine(ST_MakePointM(-350,300,0),ST_MakePointM(-410,490,0)) ↩
    ,10,20)'
-- Run the query !
SELECT track_id, dist FROM (
  SELECT track_id, ST_DistanceCPA(tr,:qt) dist
  FROM trajectories
```

```
  ORDER BY tr |=| :qt
  LIMIT 5
) foo;
 track_id        dist
----------+------------------
      395 | 0.576496831518066
      380 |  5.06797130410151
      390 |  7.72262293958322
      385 |   9.8004461358071
      405 |  10.9534397988433
(5 rows)
```

Veja também.

ST_DistanceCPA, ST_ClosestPointOfApproach, ST_IsValidTrajectory

7.10.2.3 <#>

<#> — Retorna a distância 2D entre as caixas limitadoras de A e B.

Synopsis

double precision **<#>**(geometry A , geometry B);

Descrição

O operador <#> retorna a distância entre dois pontos flutuantes, possivelmente lendo eles de um index espacial (PostgreSQL 9.1+ requerido). Útil para tornar vizinhos mais próximos **aproximar** a distância pedida.

Note
Esse operador fará uso dos indexes que podem estar disponíveis nas geometrias. É diferente de outros operadores que usam indexes espaciais em que eles só são usados quando o operador está na oração ORDEM.

Note
O index só rejeita se uma das geometrias é uma constante ex. ORDER BY (ST_GeomFromText('POINT(1 2)') <#> geom) ao invés de uma g1.geom <#>.

Disponibilidade: 2.0.0 -- KNN só está disponível para PostgreSQL 9.1+

Exemplos

```
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)
```

Veja também.

ST_DWithin, ST_Distance, <->

7.10.2.4 <<->>

<<->> — Retorna a distância n-D entre as centroides das caixas limitadoras de A e B.

Synopsis

double precision **<<->>**(geometry A , geometry B);

Descrição

O operador <<−>> retorna a distância (euclidiana) n-D entre as centroides das caixas limitadoras de duas geometrias. Útil para para tornar vizinhos mais próximos **aproximar** a distância perdida.

Note
Esse operador fará uso dos indexes n-D GiST que podem estar disponíveis nas geometrias. É diferente de outros operadores que usam indexes espaciais em que eles só são usados quando o operador está na oração ORDEM.

Note
O index só rejeita se uma das geometrias é uma constante (não em uma subquery/cte). ex. 'SRID=3005;POINT(1011102 450541)'::geometria ao invés de uma .geom

Disponibilidade: 2.2.0 -- KNN só está disponível para PostgreSQL 9.1+

Veja também.

<<#>>, <->

7.10.2.5 <<#>>

<<#>> — Retorna a distância n-D entre as caixas limitadoras de A e B.

Synopsis

double precision **<<#>>**(geometry A , geometry B);

Descrição

O operador <<#>> retorna a distância entre dois pontos flutuantes, possivelmente lendo eles de um index espacial (PostgreSQL 9.1+ requerido). Útil para tornar vizinhos mais próximos **aproximar** uma distância pedida.

Note
Esse operador fará uso dos indexes que podem estar disponíveis nas geometrias. É diferente de outros operadores que usam indexes espaciais em que eles só são usados quando o operador está na oração ORDEM.

Note
O index só rejeita se uma das geometrias é ua constante ex. ORDER BY (ST_GeomFromText('POINT(1 2)') <<#>> geom) ao invés de g1.geom <<#>>.

Disponibilidade: 2.2.0 -- KNN só está disponível para PostgreSQL 9.1+

Veja também.

<<->>, <#>

7.11 Spatial Relationships

7.11.1 Topological Relationships

7.11.1.1 ST_3DIntersects

ST_3DIntersects — Tests if two geometries spatially intersect in 3D - only for points, linestrings, polygons, polyhedral surface (area)

Synopsis

boolean **ST_3DIntersects**(geometry geomA , geometry geomB);

Description

Overlaps, Touches, Within all imply spatial intersection. If any of the aforementioned returns true, then the geometries also spatially intersect. Disjoint implies false for spatial intersection.

Note
This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries.

Changed: 3.0.0 SFCGAL backend removed, GEOS backend supports TINs.

Availability: 2.0.0

This function supports 3d and will not drop the z-index.

This function supports Polyhedral surfaces.

This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1

Geometry Examples

```
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               | t
(1 row)
```

TIN Examples

```
SELECT ST_3DIntersects('TIN(((0 0 0,1 0 0,0 1 0,0 0 0)))'::geometry, 'POINT(.1 .1 0)':: ↩
    geometry);
 st_3dintersects
-----------------
 t
```

See Also

ST_Intersects

7.11.1.2 ST_Contains

ST_Contains — Tests if every point of B lies in A, and their interiors have a point in common

Synopsis

boolean **ST_Contains**(geometry geomA, geometry geomB);

Description

Returns TRUE if geometry A contains geometry B. A contains B if and only if all points of B lie inside (i.e. in the interior or boundary of) A (or equivalently, no points of B lie in the exterior of A), and the interiors of A and B have at least one point in common.

In mathematical terms: $ST\_Contains(A, B) \Leftrightarrow (A \bigcap B = B) \wedge (Int(A) \bigcap Int(B) \neq \varnothing)$

The contains relationship is reflexive: every geometry contains itself. (In contrast, in the ST_ContainsProperly predicate a geometry does *not* properly contain itself.) The relationship is antisymmetric: if `ST_Contains(A,B) = true` and `ST_Contains(B,A) = true`, then the two geometries must be topologically equal (`ST_Equals(A,B) = true`).

ST_Contains is the converse of ST_Within. So, `ST_Contains(A,B) = ST_Within(B,A)`.

Note
Because the interiors must have a common point, a subtlety of the definition is that polygons and lines do *not* contain lines and points lying fully in their boundary. For further details see Subtleties of OGC Covers, Contains, Within. The ST_Covers predicate provides a more inclusive relationship.

Note
This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries. To avoid index use, use the function `_ST_Contains`.

Performed by the GEOS module

Enhanced: 2.3.0 Enhancement to PIP short-circuit extended to support MultiPoints with few points. Prior versions only supported point in polygon.

Important
Enhanced: 3.0.0 enabled support for `GEOMETRYCOLLECTION`

Important
Do not use this function with invalid geometries. You will get unexpected results.

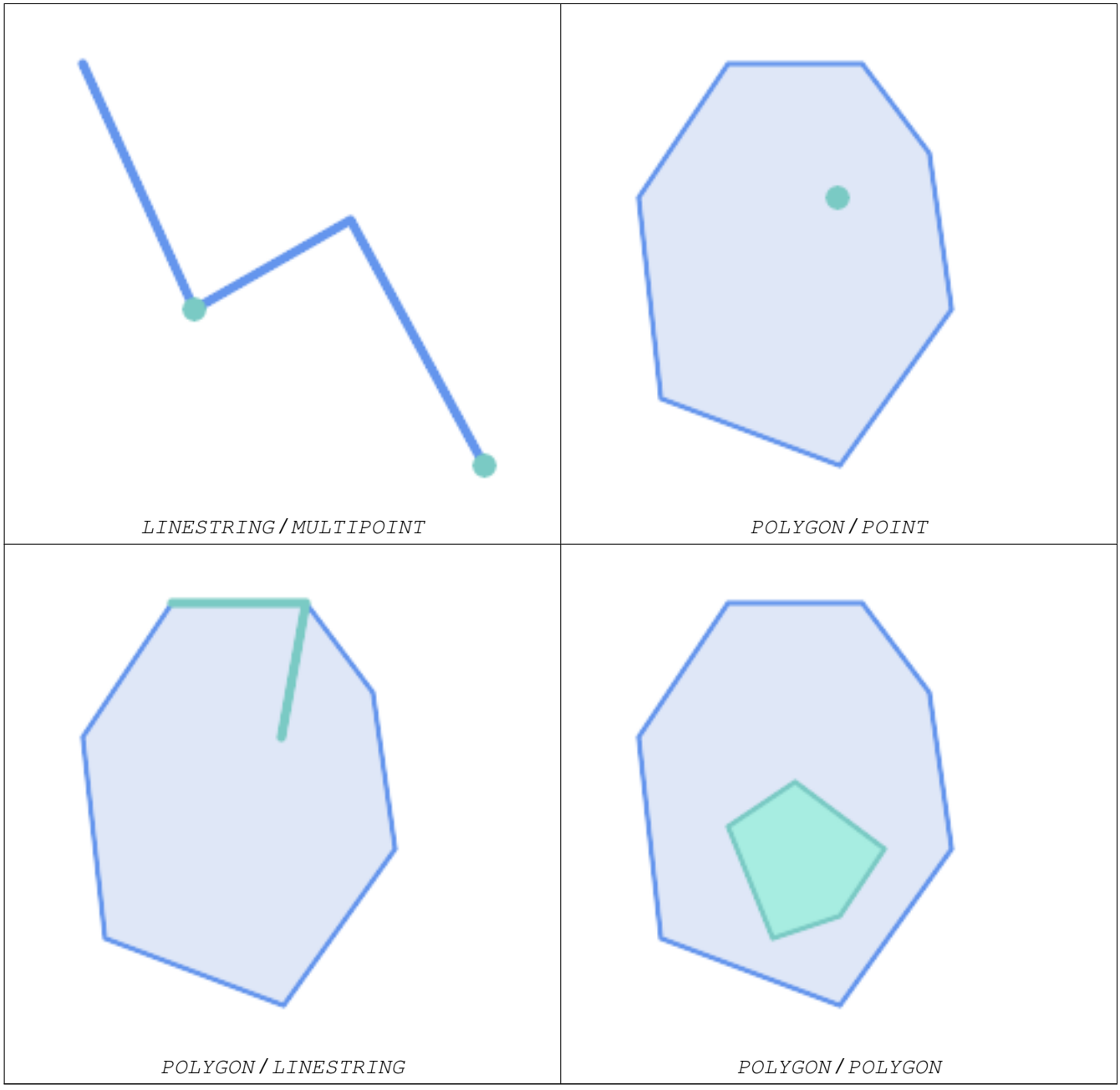
NOTE: this is the "allowable" version that returns a boolean, not an integer.

This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.2 // s2.1.13.3 - same as within(geometry B, geometry A)

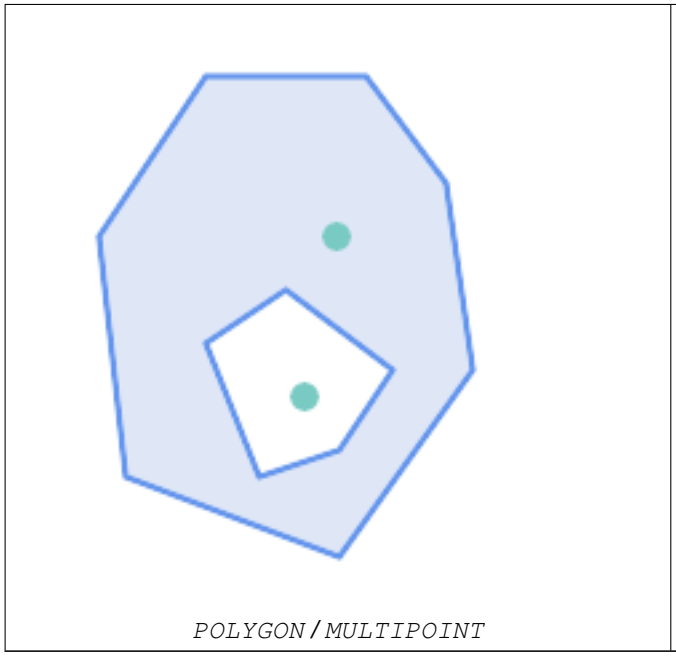
This method implements the SQL/MM specification. SQL-MM 3: 5.1.31

Examples

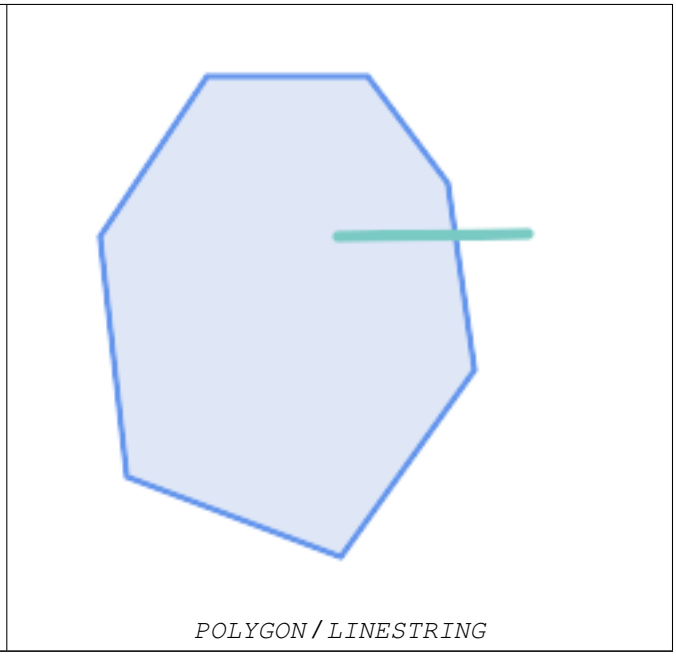
`ST_Contains` returns `TRUE` in the following situations:



ST_Contains returns FALSE in the following situations:

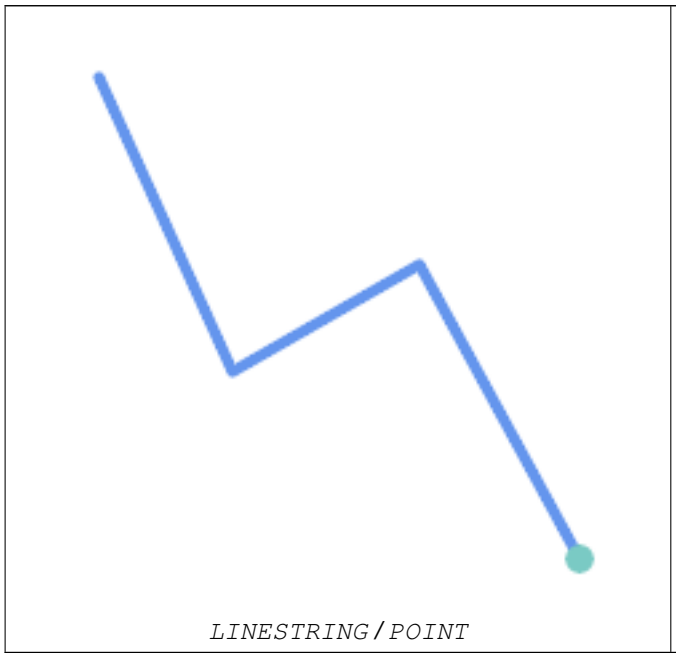


POLYGON/MULTIPOINT

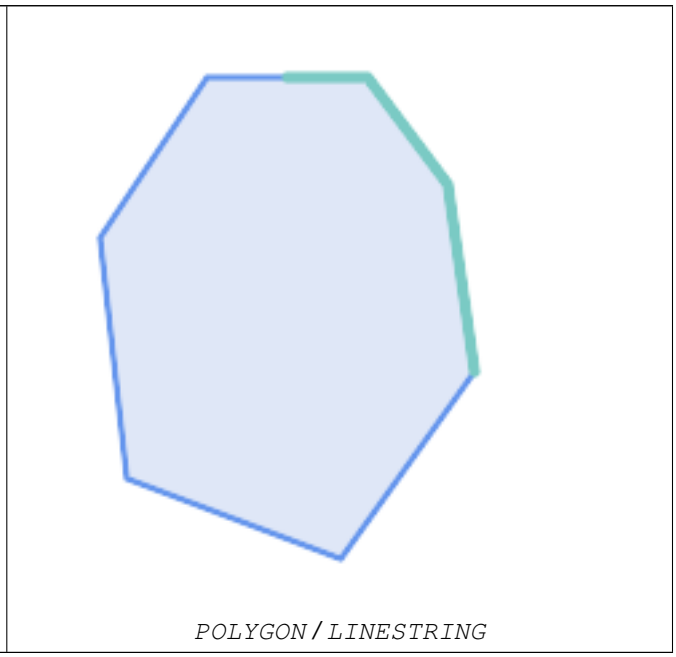


POLYGON/LINESTRING

Due to the interior intersection condition `ST_Contains` returns `FALSE` in the following situations (whereas `ST_Covers` returns `TRUE`):



LINESTRING/POINT



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
```

See Also

ST_Boundary, ST_ContainsProperly, ST_Covers, ST_CoveredBy, ST_Equals, ST_Within

7.11.1.3 ST_ContainsProperly

ST_ContainsProperly — Tests if every point of B lies in the interior of A

Synopsis

boolean **ST_ContainsProperly**(geometry geomA, geometry geomB);

Description

Returns `true` if every point of B lies in the interior of A (or equivalently, no point of B lies in the the boundary or exterior of A).

In mathematical terms: *ST_ContainsProperly(A, B)* $\Leftrightarrow$ *Int(A)* $\bigcap$ *B = B*

A contains B properly if the DE-9IM Intersection Matrix for the two geometries matches [T**FF*FF*]

A does not properly contain itself, but does contain itself.

A use for this predicate is computing the intersections of a set of geometries with a large polygonal geometry. Since intersection is a fairly slow operation, it can be more efficient to use containsProperly to filter out test geometries which lie fully inside the area. In these cases the intersection is known a priori to be exactly the original test geometry.

Note
This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries. To avoid index use, use the function `_ST_ContainsProperly`.

Note
The advantage of this predicate over ST_Contains and ST_Intersects is that it can be computed more efficiently, with no need to compute topology at individual points.

Performed by the GEOS module.

Availability: 1.4.0

Important
Enhanced: 3.0.0 enabled support for `GEOMETRYCOLLECTION`

Important
Do not use this function with invalid geometries. You will get unexpected results.

Examples

```
--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
```

See Also

ST_GeometryType, ST_Boundary, ST_Contains, ST_Covers, ST_CoveredBy, ST_Equals, ST_Relate, ST_Within

7.11.1.4 ST_CoveredBy

ST_CoveredBy — Tests if every point of A lies in B

Synopsis

boolean **ST_CoveredBy**(geometry geomA, geometry geomB);
boolean **ST_CoveredBy**(geography geogA, geography geogB);

Description

Returns `true` if every point in Geometry/Geography A lies inside (i.e. intersects the interior or boundary of) Geometry/Geography B. Equivalently, tests that no point of A lies outside (in the exterior of) B.

In mathematical terms: *ST_CoveredBy(A, B)* $\Leftrightarrow A \bigcap B = A$

ST_CoveredBy is the converse of ST_Covers. So, `ST_CoveredBy(A,B) = ST_Covers(B,A)`.

Generally this function should be used instead of ST_Within, since it has a simpler definition which does not have the quirk that "boundaries are not within their geometry".

Note
This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries. To avoid index use, use the function `_ST_CoveredBy`.

Important
Enhanced: 3.0.0 enabled support for `GEOMETRYCOLLECTION`

Important
Do not use this function with invalid geometries. You will get unexpected results.

Performed by the GEOS module

Availability: 1.2.2

NOTE: this is the "allowable" version that returns a boolean, not an integer.

Not an OGC standard, but Oracle has it too.

Examples

```
--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)
```

See Also

ST_Contains, ST_Covers, ST_ExteriorRing, ST_Within

7.11.1.5 ST_Covers

ST_Covers — Tests if every point of B lies in A

Synopsis

boolean **ST_Covers**(geometry geomA, geometry geomB);
boolean **ST_Covers**(geography geogpolyA, geography geogpointB);

Description

Returns `true` if every point in Geometry/Geography B lies inside (i.e. intersects the interior or boundary of) Geometry/Geography A. Equivalently, tests that no point of B lies outside (in the exterior of) A.

In mathematical terms: *ST_Covers(A, B)* $\Leftrightarrow A \bigcap B = B$

ST_Covers is the converse of ST_CoveredBy. So, `ST_Covers(A,B) = ST_CoveredBy(B,A)`.

Generally this function should be used instead of ST_Contains, since it has a simpler definition which does not have the quirk that "geometries do not contain their boundary".

Note
This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries. To avoid index use, use the function `_ST_Covers`.

Important
Enhanced: 3.0.0 enabled support for `GEOMETRYCOLLECTION`

Important
Do not use this function with invalid geometries. You will get unexpected results.

Performed by the GEOS module

Enhanced: 2.4.0 Support for polygon in polygon and line in polygon added for geography type

Enhanced: 2.3.0 Enhancement to PIP short-circuit for geometry extended to support MultiPoints with few points. Prior versions only supported point in polygon.

Availability: 1.5 - support for geography was introduced.

Availability: 1.2.2

NOTE: this is the "allowable" version that returns a boolean, not an integer.

Not an OGC standard, but Oracle has it too.

Examples

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
```

See Also

ST_Contains, ST_CoveredBy, ST_Within

7.11.1.6 ST_Crosses

ST_Crosses — Tests if two geometries have some, but not all, interior points in common

Synopsis

boolean **ST_Crosses**(geometry g1, geometry g2);

Description

Compares two geometry objects and returns `true` if their intersection "spatially crosses"; that is, the geometries have some, but not all interior points in common. The intersection of the interiors of the geometries must be non-empty and must have dimension less than the maximum dimension of the two input geometries, and the intersection of the two geometries must not equal either geometry. Otherwise, it returns `false`. The crosses relation is symmetric and irreflexive.

In mathematical terms: $ST\_Crosses(A, B) \Leftrightarrow (dim(\ Int(A) \bigcap Int(B)\ ) < max(\ dim(\ Int(A)\ ), dim(\ Int(B)\ )\ )) \wedge (A \bigcap B \neq A) \wedge (A \bigcap B \neq B)$

Geometries cross if their DE-9IM Intersection Matrix matches:

- `T*T******` for Point/Line, Point/Area, and Line/Area situations
- `T*****T**` for Line/Point, Area/Point, and Area/Line situations

- `0********` for Line/Line situations
- the result is `false` for Point/Point and Area/Area situations

Note
The OpenGIS Simple Features Specification defines this predicate only for Point/Line, Point/Area, Line/Line, and Line/Area situations. JTS / GEOS extends the definition to apply to Line/Point, Area/Point and Area/Line situations as well. This makes the relation symmetric.

Note
This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries.

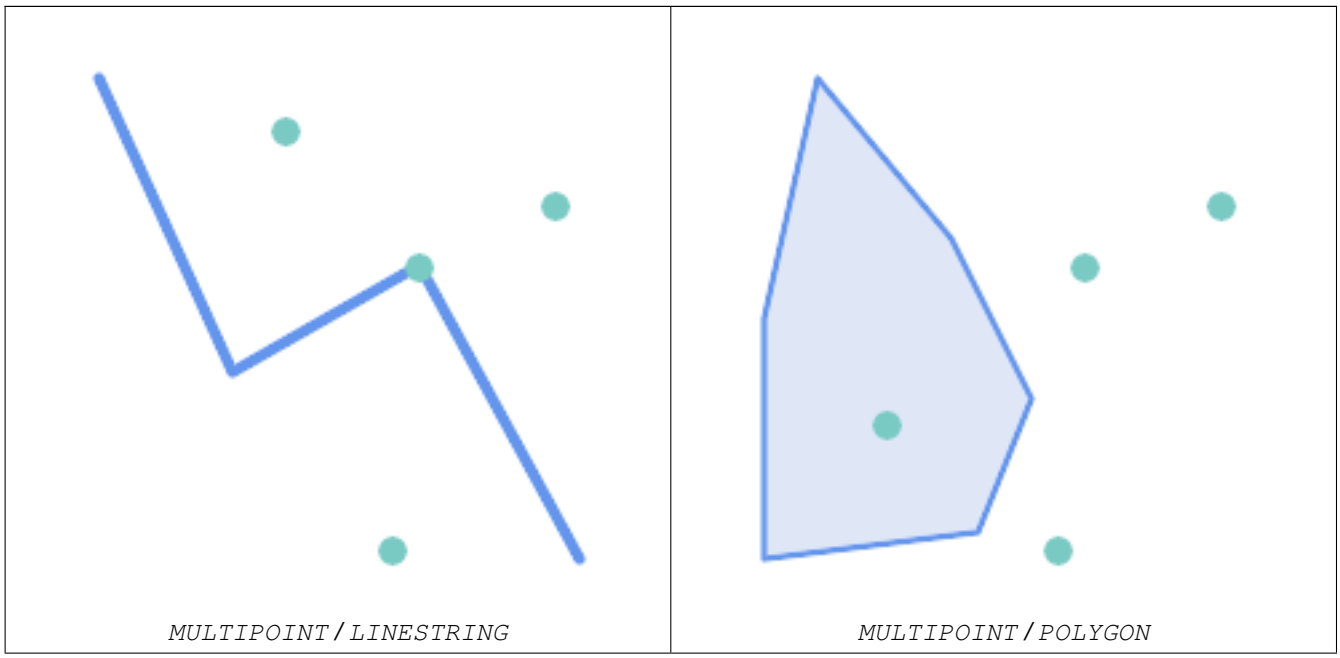
Important
Enhanced: 3.0.0 enabled support for `GEOMETRYCOLLECTION`

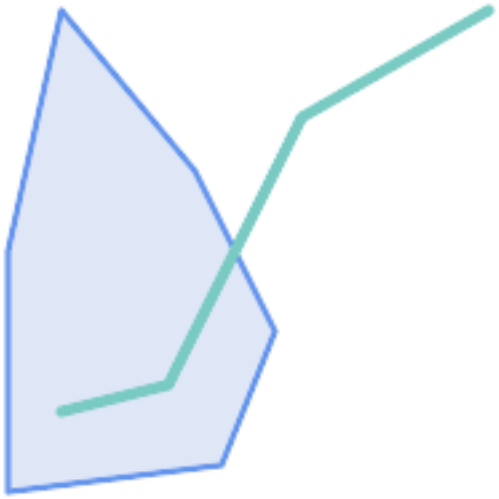
This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.13.3

This method implements the SQL/MM specification. SQL-MM 3: 5.1.29

Examples

The following situations all return `true`.





LINESTRING/POLYGON

LINESTRING/LINESTRING

Consider a situation where a user has two tables: a table of roads and a table of highways.

```
CREATE TABLE roads (
  id serial NOT NULL,
  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)
);
```

To determine a list of roads that cross a highway, use a query similiar to:

```
SELECT roads.id
FROM roads, highways
WHERE ST_Crosses(roads.geom, highways.geom);
```

See Also

ST_Contains, ST_Overlaps

7.11.1.7 ST_Disjoint

ST_Disjoint — Tests if two geometries have no points in common

Synopsis

boolean **ST_Disjoint**(geometry A , geometry B);

Description

Returns `true` if two geometries are disjoint. Geometries are disjoint if they have no point in common.

If any other spatial relationship is true for a pair of geometries, they are not disjoint. Disjoint implies that ST_Intersects is false.

In mathematical terms: *ST_Disjoint(A, B)* $\Leftrightarrow A \bigcap B = \varnothing$

Important
Enhanced: 3.0.0 enabled support for `GEOMETRYCOLLECTION`

Performed by the GEOS module

Note
This function call does not use indexes. A negated ST_Intersects predicate can be used as a more performant alternative that uses indexes: `ST_Disjoint(A,B) = NOT ST_Intersects(A,B)`

Note
NOTE: this is the "allowable" version that returns a boolean, not an integer.

This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.2 //s2.1.13.3 - a.Relate(b, 'FF*FF****')

This method implements the SQL/MM specification. SQL-MM 3: 5.1.26

Examples

```
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)
```

See Also

ST_Intersects

7.11.1.8 ST_Equals

ST_Equals — Tests if two geometries include the same set of points

Synopsis

boolean **ST_Equals**(geometry A, geometry B);

Description

Returns `true` if the given geometries are "topologically equal". Use this for a 'better' answer than '='. Topological equality means that the geometries have the same dimension, and their point-sets occupy the same space. This means that the order of vertices may be different in topologically equal geometries. To verify the order of points is consistent use ST_OrderingEquals (it must be noted ST_OrderingEquals is a little more stringent than simply verifying order of points are the same).

In mathematical terms: *ST_Equals(A, B)* $\Leftrightarrow$ *A = B*

The following relation holds: *ST_Equals(A, B)* $\Leftrightarrow$ *ST_Within(A,B)* $\wedge$ *ST_Within(B,A)*

Important
Enhanced: 3.0.0 enabled support for `GEOMETRYCOLLECTION`


This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.2


This method implements the SQL/MM specification. SQL-MM 3: 5.1.24

Changed: 2.2.0 Returns true even for invalid geometries if they are binary equal

Examples

```
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)
```

See Also

ST_IsValid, ST_OrderingEquals, ST_Reverse, ST_Within

7.11.1.9 ST_Intersects

ST_Intersects — Tests if two geometries intersect (they have at least one point in common)

Synopsis

boolean **ST_Intersects**(geometry geomA , geometry geomB);
boolean **ST_Intersects**(geography geogA , geography geogB);

Description

Returns `true` if two geometries intersect. Geometries intersect if they have any point in common.

For geography, a distance tolerance of 0.00001 meters is used (so points that are very close are considered to intersect).

In mathematical terms: *ST_Intersects(A, B)* $\Leftrightarrow A \bigcap B \neq \varnothing$

Geometries intersect if their DE-9IM Intersection Matrix matches one of:

- `T********`
- `*T*******`
- `***T*****`
- `****T****`

Spatial intersection is implied by all the other spatial relationship tests, except ST_Disjoint, which tests that geometries do NOT intersect.

Note
This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries.

Changed: 3.0.0 SFCGAL version removed and native support for 2D TINS added.

Enhanced: 2.5.0 Supports GEOMETRYCOLLECTION.

Enhanced: 2.3.0 Enhancement to PIP short-circuit extended to support MultiPoints with few points. Prior versions only supported point in polygon.

Performed by the GEOS module (for geometry), geography is native

Availability: 1.5 support for geography was introduced.

Note
For geography, this function has a distance tolerance of about 0.00001 meters and uses the sphere rather than spheroid calculation.

Note
NOTE: this is the "allowable" version that returns a boolean, not an integer.

This method implements the OGC 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))

This method implements the SQL/MM specification. SQL-MM 3: 5.1.27

This method supports Circular Strings and Curves.

This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

Geometry Examples

```
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)

-- Look up in table. Make sure table has a GiST index on geometry column for faster lookup.
SELECT id, name FROM cities WHERE ST_Intersects(geom, 'SRID=4326;POLYGON((28 53,27.707  ↩
    52.293,27 52,26.293 52.293,26 53,26.293 53.707,27 54,27.707 53.707,28 53))');
 id | name
----+-------
  2 | Minsk
(1 row)
```

Geography Examples

```
SELECT ST_Intersects(
    'SRID=4326;LINESTRING(-43.23456 72.4567,-43.23456 72.4568)'::geography,
    'SRID=4326;POINT(-43.23456 72.4567772)'::geography
    );

 st_intersects
---------------
t
```

See Also

&&, ST_3DIntersects, ST_Disjoint

7.11.1.10 ST_LineCrossingDirection

ST_LineCrossingDirection — Returns a number indicating the crossing behavior of two LineStrings

Synopsis

integer **ST_LineCrossingDirection**(geometry linestringA, geometry linestringB);

Description

Given two linestrings returns an integer between -3 and 3 indicating what kind of crossing behavior exists between them. 0 indicates no crossing. This is only supported for `LINESTRING`s.

The crossing number has the following meaning:

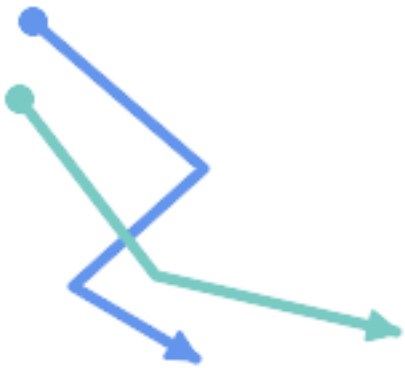
- 0: LINE NO CROSS
- -1: LINE CROSS LEFT
- 1: LINE CROSS RIGHT

- -2: LINE MULTICROSS END LEFT
- 2: LINE MULTICROSS END RIGHT
- -3: LINE MULTICROSS END SAME FIRST LEFT
- 3: LINE MULTICROSS END SAME FIRST RIGHT

Availability: 1.4

Examples

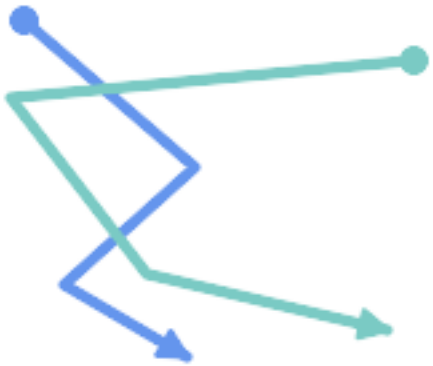
Example: LINE CROSS LEFT and LINE CROSS RIGHT



Blue: Line A; Green: Line B

```
SELECT ST_LineCrossingDirection(lineA, lineB) As A_cross_B,
       ST_LineCrossingDirection(lineB, lineA) As B_cross_A
FROM (SELECT
  ST_GeomFromText('LINESTRING(25 169,89 114,40 70,86 43)') As lineA,
  ST_GeomFromText('LINESTRING (20 140, 71 74, 161 53)') As lineB
  ) As foo;

 A_cross_B | B_cross_A
-----------+-----------
        -1 |         1
```

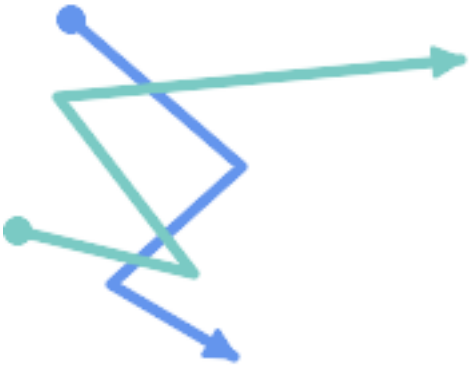
Example: LINE MULTICROSS END SAME FIRST LEFT and LINE MULTICROSS END SAME FIRST RIGHT



Blue: Line A; Green: Line B

```
SELECT ST_LineCrossingDirection(lineA, lineB) As A_cross_B,
       ST_LineCrossingDirection(lineB, lineA) As B_cross_A
FROM (SELECT
 ST_GeomFromText('LINESTRING(25 169,89 114,40 70,86 43)') As lineA,
 ST_GeomFromText('LINESTRING(171 154,20 140,71 74,161 53)') As lineB
  ) As foo;

 A_cross_B | B_cross_A
-----------+-----------
         3 |        -3
```

Example: LINE MULTICROSS END LEFT and LINE MULTICROSS END RIGHT



Blue: Line A; Green: Line B

```
SELECT ST_LineCrossingDirection(lineA, lineB) As A_cross_B,
       ST_LineCrossingDirection(lineB, lineA) As B_cross_A
FROM (SELECT
  ST_GeomFromText('LINESTRING(25 169,89 114,40 70,86 43)') As lineA,
  ST_GeomFromText('LINESTRING(5 90, 71 74, 20 140, 171 154)') As lineB
  ) As foo;
```

```
 A_cross_B | B_cross_A
-----------+-----------
        -2 |         2
```

Example: Finds all streets that cross

```
SELECT s1.gid, s2.gid, ST_LineCrossingDirection(s1.geom, s2.geom)
  FROM streets s1 CROSS JOIN streets s2
         ON (s1.gid != s2.gid AND s1.geom && s2.geom )
WHERE ST_LineCrossingDirection(s1.geom, s2.geom)
> 0;
```

See Also

ST_Crosses

7.11.1.11 ST_OrderingEquals

ST_OrderingEquals — Tests if two geometries represent the same geometry and have points in the same directional order

Synopsis

boolean **ST_OrderingEquals**(geometry A, geometry B);

Description

ST_OrderingEquals compares two geometries and returns t (TRUE) if the geometries are equal and the coordinates are in the same order; otherwise it returns f (FALSE).

Note
This function is implemented as per the ArcSDE SQL specification rather than SQL-MM. http://edndoc.esri.com/arcsde/9.1/sql_api/sqlapi3.htm#ST_OrderingEquals

This method implements the SQL/MM specification. SQL-MM 3: 5.1.43

Examples

```
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)
```

See Also

&&, ST_Equals, ST_Reverse

7.11.1.12 ST_Overlaps

ST_Overlaps — Tests if two geometries have the same dimension and intersect, but each has at least one point not in the other

Synopsis

boolean **ST_Overlaps**(geometry A, geometry B);

Description

Returns TRUE if geometry A and B "spatially overlap". Two geometries overlap if they have the same dimension, their interiors intersect in that dimension. and each has at least one point inside the other (or equivalently, neither one covers the other). The overlaps relation is symmetric and irreflexive.

In mathematical terms: $ST\_Overlaps(A, B) \Leftrightarrow (dim(A) = dim(B) = dim(Int(A) \bigcap Int(B))) \wedge (A \bigcap B \neq A) \wedge (A \bigcap B \neq B)$

Note
This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries. To avoid index use, use the function `_ST_Overlaps`.

Performed by the GEOS module

Important
Enhanced: 3.0.0 enabled support for `GEOMETRYCOLLECTION`

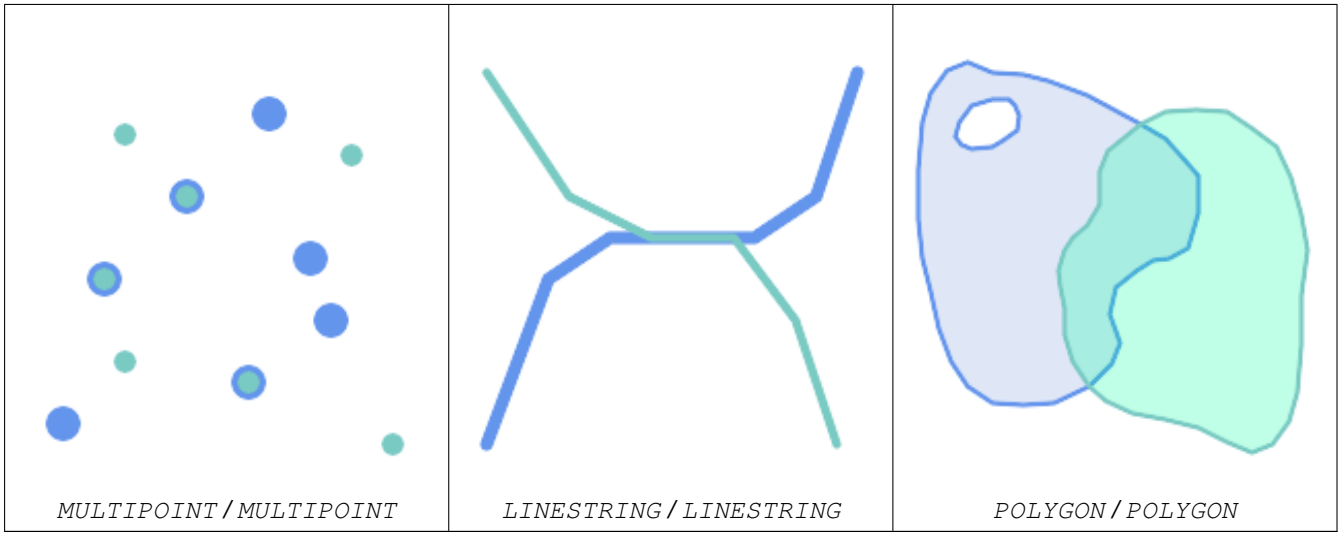
NOTE: this is the "allowable" version that returns a boolean, not an integer.

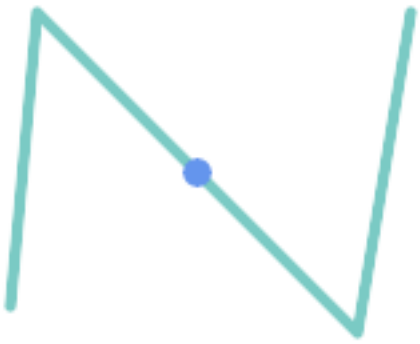

This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.2 // s2.1.13.3


This method implements the SQL/MM specification. SQL-MM 3: 5.1.32

Examples

`ST_Overlaps` returns `TRUE` in the following situations:



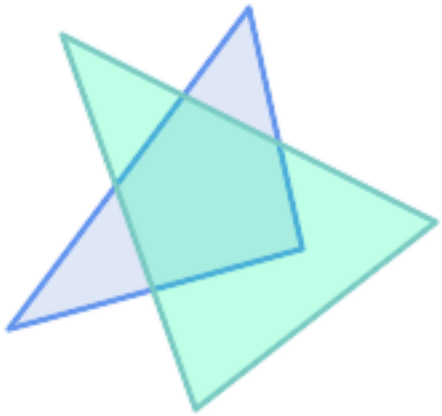


A Point on a LineString is contained, but since it has lower dimension it does not overlap or cross.

```
SELECT ST_Overlaps(a,b) AS overlaps,       ST_Crosses(a,b) AS crosses,
       ST_Intersects(a, b) AS intersects,  ST_Contains(b,a) AS b_contains_a
FROM (SELECT ST_GeomFromText('POINT (100 100)') As a,
             ST_GeomFromText('LINESTRING (30 50, 40 160, 160 40, 180 160)')  AS b) AS t

overlaps | crosses | intersects | b_contains_a
---------+---------------------+--------------
f        | f       | t          | t
```

A LineString that partly covers a Polygon intersects and crosses, but does not overlap since it has different dimension.

```
SELECT ST_Overlaps(a,b) AS overlaps,        ST_Crosses(a,b) AS crosses,
       ST_Intersects(a, b) AS intersects,   ST_Contains(a,b) AS contains
FROM (SELECT ST_GeomFromText('POLYGON ((40 170, 90 30, 180 100, 40 170))') AS a,
             ST_GeomFromText('LINESTRING(10 10, 190 190)') AS b) AS t;

 overlap | crosses | intersects | contains
---------+---------+------------+--------------
 f       | t       | t          | f
```



Two Polygons that intersect but with neither contained by the other overlap, but do not cross because their intersection has the same dimension.

```
SELECT ST_Overlaps(a,b) AS overlaps,       ST_Crosses(a,b) AS crosses,
       ST_Intersects(a, b) AS intersects,  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 dim_int
FROM (SELECT ST_GeomFromText('POLYGON ((40 170, 90 30, 180 100, 40 170))') AS a,
             ST_GeomFromText('POLYGON ((110 180, 20 60, 130 90, 110 180))') AS b) As t;

 overlaps | crosses | intersects | b_contains_a | dim_a | dim_b | dim_int
```

```
----------+---------+------------+--------------+-------+-------+-----------
 t        | f       | t          | f            |     2 |     2 |         2
```

See Also

ST_Contains, ST_Crosses, ST_Dimension, ST_Intersects

7.11.1.13 ST_Relate

ST_Relate — Tests if two geometries have a topological relationship matching an Intersection Matrix pattern, or computes their Intersection Matrix

Synopsis

boolean **ST_Relate**(geometry geomA, geometry geomB, text intersectionMatrixPattern);
text **ST_Relate**(geometry geomA, geometry geomB);
text **ST_Relate**(geometry geomA, geometry geomB, integer boundaryNodeRule);

Description

These functions allow testing and evaluating the spatial (topological) relationship between two geometries, as defined by the Dimensionally Extended 9-Intersection Model (DE-9IM).

The DE-9IM is specified as a 9-element matrix indicating the dimension of the intersections between the Interior, Boundary and Exterior of two geometries. It is represented by a 9-character text string using the symbols 'F', '0', '1', '2' (e.g. `'FF1FF0102'`).

A specific kind of spatial relationship can be tested by matching the intersection matrix to an *intersection matrix pattern*. Patterns can include the additional symbols 'T' (meaning "intersection is non-empty") and '*' (meaning "any value"). Common spatial relationships are provided by the named functions ST_Contains, ST_ContainsProperly, ST_Covers, ST_CoveredBy, ST_Crosses, ST_Disjoint, ST_Equals, ST_Intersects, ST_Overlaps, ST_Touches, and ST_Within. Using an explicit pattern allows testing multiple conditions of intersects, crosses, etc in one step. It also allows testing spatial relationships which do not have a named spatial relationship function. For example, the relationship "Interior-Intersects" has the DE-9IM pattern `T********`, which is not evaluated by any named predicate.

For more information refer to Section 5.1.

Variant 1: Tests if two geometries are spatially related according to the given `intersectionMatrixPattern`.

Note
Unlike most of the named spatial relationship predicates, this does NOT automatically include an index call. The reason is that some relationships are true for geometries which do NOT intersect (e.g. Disjoint). If you are using a relationship pattern that requires intersection, then include the && index call.

Note
It is better to use a named relationship function if available, since they automatically use a spatial index where one exists. Also, they may implement performance optimizations which are not available with full relate evalation.

Variant 2: Returns the DE-9IM matrix string for the spatial relationship between the two input geometries. The matrix string can be tested for matching a DE-9IM pattern using ST_RelateMatch.

Variant 3: Like variant 2, but allows specifying a **Boundary Node Rule**. A boundary node rule allows finer control over whether the endpoints of MultiLineStrings are considered to lie in the DE-9IM Interior or Boundary. The `boundaryNodeRule` values are:

- `1`: **OGC-Mod2** - line endpoints are in the Boundary if they occur an odd number of times. This is the rule defined by the OGC SFS standard, and is the default for `ST_Relate`.
- `2`: **Endpoint** - all endpoints are in the Boundary.
- `3`: **MultivalentEndpoint** - endpoints are in the Boundary if they occur more than once. In other words, the boundary is all the "attached" or "inner" endpoints (but not the "unattached/outer" ones).
- `4`: **MonovalentEndpoint** - endpoints are in the Boundary if they occur only once. In other words, the boundary is all the "unattached" or "outer" endpoints.

This function is not in the OGC spec, but is implied. see s2.1.13.2

This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.2 // s2.1.13.3

This method implements the SQL/MM specification. SQL-MM 3: 5.1.25

Performed by the GEOS module

Enhanced: 2.0.0 - added support for specifying boundary node rule.

Important
Enhanced: 3.0.0 enabled support for `GEOMETRYCOLLECTION`

Examples

Using the boolean-valued function to test spatial relationships.

```
SELECT ST_Relate('POINT(1 2)', ST_Buffer( 'POINT(1 2)', 2), '0FFFFF212');
st_relate
-----------
t

SELECT ST_Relate(POINT(1 2)', ST_Buffer( 'POINT(1 2)', 2), '*FF*FF212');
st_relate
-----------
t
```

Testing a custom spatial relationship pattern as a query condition, with `&&` to enable using a spatial index.

```
-- Find compounds that properly intersect (not just touch) a poly (Interior Intersects)

SELECT c.* , p.name As poly_name
    FROM polys AS p
    INNER JOIN compounds As c
          ON c.geom && p.geom
             AND ST_Relate(p.geom, c.geom,'T********');
```

Computing the intersection matrix for spatial relationships.

```
SELECT ST_Relate( 'POINT(1 2)',
                  ST_Buffer( 'POINT(1 2)', 2));
-----------
0FFFFF212

SELECT ST_Relate( 'LINESTRING(1 2, 3 4)',
                  'LINESTRING(5 6, 7 8)' );
-----------
FF1FF0102
```

Using different Boundary Node Rules to compute the spatial relationship between a LineString and a MultiLineString with a duplicate endpoint `(3 3)`:

- Using the **OGC-Mod2** rule (1) the duplicate endpoint is in the **interior** of the MultiLineString, so the DE-9IM matrix entry [aB:bI] is `0` and [aB:bB] is `F`.
- Using the **Endpoint** rule (2) the duplicate endpoint is in the **boundary** of the MultiLineString, so the DE-9IM matrix entry [aB:bI] is `F` and [aB:bB] is `0`.

```
WITH data AS (SELECT
  'LINESTRING(1 1, 3 3)'::geometry AS a_line,
  'MULTILINESTRING((3 3, 3 5), (3 3, 5 3))':: geometry AS b_multiline
)
SELECT ST_Relate( a_line, b_multiline, 1) AS bnr_mod2,
       ST_Relate( a_line, b_multiline, 2) AS bnr_endpoint
    FROM data;

 bnr_mod2  | bnr_endpoint
-----------+--------------
 FF10F0102 | FF1F00102
```

See Also

Section 5.1, ST_RelateMatch, ST_Contains, ST_ContainsProperly, ST_Covers, ST_CoveredBy, ST_Crosses, ST_Disjoint, ST_Equals, ST_Intersects, ST_Overlaps, ST_Touches, ST_Within

7.11.1.14 ST_RelateMatch

ST_RelateMatch — Tests if a DE-9IM Intersection Matrix matches an Intersection Matrix pattern

Synopsis

boolean **ST_RelateMatch**(text intersectionMatrix, text intersectionMatrixPattern);

Description

Tests if a Dimensionally Extended 9-Intersection Model (DE-9IM) `intersectionMatrix` value satisfies an `intersectionMatr`
Intersection matrix values can be computed by ST_Relate.

For more information refer to Section 5.1.

Performed by the GEOS module

Availability: 2.0.0

Examples

```
SELECT ST_RelateMatch('101202FFF', 'TTTTTTFFF') ;
-- result --
t
```

Patterns for common spatial relationships matched against intersection matrix values, for a line in various positions relative to a polygon

```
SELECT pat.name AS relationship, pat.val AS pattern,
       mat.name AS position, mat.val AS matrix,
       ST_RelateMatch(mat.val, pat.val) AS match
    FROM (VALUES ( 'Equality', 'T1FF1FFF1' ),
                 ( 'Overlaps', 'T*T***T**' ),
                 ( 'Within',   'T*F**F***' ),
                 ( 'Disjoint', 'FF*FF****' )) AS pat(name,val)
    CROSS JOIN
        (VALUES  ('non-intersecting', 'FF1FF0212'),
                 ('overlapping',      '1010F0212'),
                 ('inside',           '1FF0FF212')) AS mat(name,val);

 relationship |  pattern  |     position     |  matrix   | match
--------------+-----------+------------------+-----------+-------
 Equality     | T1FF1FFF1 | non-intersecting | FF1FF0212 | f
 Equality     | T1FF1FFF1 | overlapping      | 1010F0212 | f
 Equality     | T1FF1FFF1 | inside           | 1FF0FF212 | f
 Overlaps     | T*T***T** | non-intersecting | FF1FF0212 | f
 Overlaps     | T*T***T** | overlapping      | 1010F0212 | t
 Overlaps     | T*T***T** | inside           | 1FF0FF212 | f
 Within       | T*F**F*** | non-intersecting | FF1FF0212 | f
 Within       | T*F**F*** | overlapping      | 1010F0212 | f
 Within       | T*F**F*** | inside           | 1FF0FF212 | t
 Disjoint     | FF*FF**** | non-intersecting | FF1FF0212 | t
 Disjoint     | FF*FF**** | overlapping      | 1010F0212 | f
 Disjoint     | FF*FF**** | inside           | 1FF0FF212 | f
```

See Also

Section 5.1, ST_Relate

7.11.1.15 ST_Touches

ST_Touches — Tests if two geometries have at least one point in common, but their interiors do not intersect

Synopsis

boolean **ST_Touches**(geometry A, geometry B);

Description

Returns `TRUE` if A and B intersect, but their interiors do not intersect. Equivalently, A and B have at least one point in common, and the common points lie in at least one boundary. For Point/Point inputs the relationship is always `FALSE`, since points do not have a boundary.

In mathematical terms: $ST\_Touches(A, B) \Leftrightarrow (Int(A) \bigcap Int(B) \neq \varnothing) \wedge (A \bigcap B \neq \varnothing)$

This relationship holds if the DE-9IM Intersection Matrix for the two geometries matches one of:

- FT*******
- F**T*****
- F***T****

Note
This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries. To avoid using an index, use `_ST_Touches` instead.

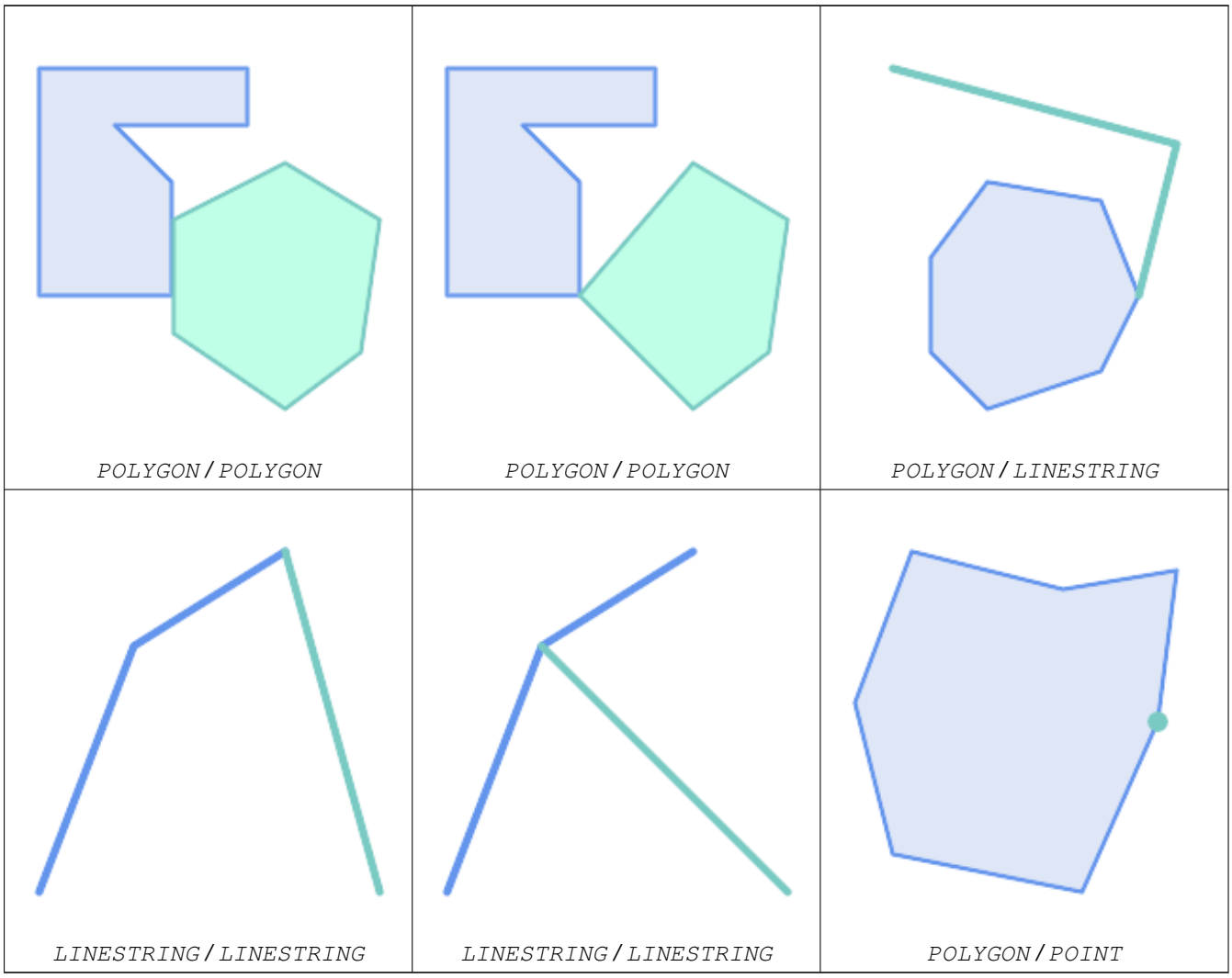
Important
Enhanced: 3.0.0 enabled support for `GEOMETRYCOLLECTION`

This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.2 // s2.1.13.3

This method implements the SQL/MM specification. SQL-MM 3: 5.1.28

Examples

The `ST_Touches` predicate returns `TRUE` in the following examples.




```
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)
```

7.11.1.16 ST_Within

ST_Within — Tests if every point of A lies in B, and their interiors have a point in common

Synopsis

boolean **ST_Within**(geometry A, geometry B);

Description

Returns TRUE if geometry A is within geometry B. A is within B if and only if all points of A lie inside (i.e. in the interior or boundary of) B (or equivalently, no points of A lie in the exterior of B), and the interiors of A and B have at least one point in common.

For this function to make sense, the source geometries must both be of the same coordinate projection, having the same SRID.

In mathematical terms: $ST\_Within(A, B) \Leftrightarrow (A \bigcap B = A) \wedge (Int(A) \bigcap Int(B) \neq \varnothing)$

The within relation is reflexive: every geometry is within itself. The relation is antisymmetric: if `ST_Within(A,B) = true` and `ST_Within(B,A) = true`, then the two geometries must be topologically equal (`ST_Equals(A,B) = true`).

ST_Within is the converse of ST_Contains. So, `ST_Within(A,B) = ST_Contains(B,A)`.

Note
Because the interiors must have a common point, a subtlety of the definition is that lines and points lying fully in the boundary of polygons or lines are *not* within the geometry. For further details see Subtleties of OGC Covers, Contains, Within. The ST_CoveredBy predicate provides a more inclusive relationship.

Note
This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries. To avoid index use, use the function `_ST_Within`.

Performed by the GEOS module

Enhanced: 2.3.0 Enhancement to PIP short-circuit for geometry extended to support MultiPoints with few points. Prior versions only supported point in polygon.

Important
Enhanced: 3.0.0 enabled support for `GEOMETRYCOLLECTION`

Important
Do not use this function with invalid geometries. You will get unexpected results.

NOTE: this is the "allowable" version that returns a boolean, not an integer.

 This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.2 // s2.1.13.3 - a.Relate(b, 'T*F**F***')

 This method implements the SQL/MM specification. SQL-MM 3: 5.1.30

Examples

```
--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)
```

See Also

ST_Contains, ST_CoveredBy, ST_Equals, ST_IsValid

7.11.2 Distance Relationships

7.11.2.1 ST_3DDWithin

ST_3DDWithin — Tests if two 3D geometries are within a given 3D distance

Synopsis

boolean **ST_3DDWithin**(geometry g1, geometry g2, double precision distance_of_srid);

Description

Returns true if the 3D distance between two geometry values is no larger than distance `distance_of_srid`. The distance is specified in units defined by the spatial reference system of the geometries. For this function to make sense the source geometries must be in the same coordinate system (have the same SRID).

Note
This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries.

 This function supports 3d and will not drop the z-index.

 This function supports Polyhedral surfaces.

 This method implements the SQL/MM specification. SQL-MM ?

Availability: 2.0.0

Examples

```
-- Geometry example - units in meters (SRID: 2163 US National Atlas Equal area) (3D point  ↩
    and line compared 2D point and line)
-- Note: currently no vertical datum support so Z is not transformed and assumed to be same ↩
     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 42.1546 ↩
          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 42.1546 ↩
          20)'),2163),
      126.8
    ) As within_dist_2d;

 within_dist_3d | within_dist_2d
----------------+----------------
 f              | t
```

See Also

ST_3DDFullyWithin, ST_DWithin, ST_DFullyWithin, ST_3DDistance, ST_Distance, ST_3DMaxDistance, ST_Transform

7.11.2.2 ST_3DDFullyWithin

ST_3DDFullyWithin — Tests if two 3D geometries are entirely within a given 3D distance

Synopsis

boolean **ST_3DDFullyWithin**(geometry g1, geometry g2, double precision distance);

Description

Returns true if the 3D geometries are fully within the specified distance of one another. The distance is specified in units defined by the spatial reference system of the geometries. For this function to make sense, the source geometries must both be of the same coordinate projection, having the same SRID.

Note
This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries.

Availability: 2.0.0

 This function supports 3d and will not drop the z-index.

 This function supports Polyhedral surfaces.

Examples

```
-- 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
```

See Also

ST_3DDWithin, ST_DWithin, ST_DFullyWithin, ST_3DMaxDistance

7.11.2.3 ST_DFullyWithin

ST_DFullyWithin — Tests if two geometries are entirely within a given distance

Synopsis

boolean **ST_DFullyWithin**(geometry g1, geometry g2, double precision distance);

Description

Returns true if the geometries are entirely within the specified distance of one another. The distance is specified in units defined by the spatial reference system of the geometries. For this function to make sense, the source geometries must both be of the same coordinate projection, having the same SRID.

Note
This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries.

Availability: 1.5.0

Examples

```
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              |
```

See Also

ST_MaxDistance, ST_DWithin, ST_3DDWithin, ST_3DDFullyWithin

7.11.2.4 ST_DWithin

ST_DWithin — Tests if two geometries are within a given distance

Synopsis

boolean **ST_DWithin**(geometry g1, geometry g2, double precision distance_of_srid);
boolean **ST_DWithin**(geography gg1, geography gg2, double precision distance_meters, boolean use_spheroid = true);

Description

Returns true if the geometries are within a given distance

For geometry: The distance is specified in units defined by the spatial reference system of the geometries. For this function to make sense, the source geometries must be in the same coordinate system (have the same SRID).

For geography: units are in meters and distance measurement defaults to `use_spheroid = true`. For faster evaluation use `use_spheroid = false` to measure on the sphere.

Note
Use ST_3DDWithin for 3D geometries.

Note
This function call includes a bounding box comparison that makes use of any indexes that are available on the geometries.

 This method implements the OGC Simple Features Implementation Specification for SQL 1.1.

Availability: 1.5.0 support for geography was introduced

Enhanced: 2.1.0 improved speed for geography. See Making Geography faster for details.

Enhanced: 2.1.0 support for curved geometries was introduced.

Prior to 1.3, ST_Expand was commonly used in conjunction with && and ST_Distance to test for distance, and in pre-1.3.4 this function used that logic. From 1.3.4, ST_DWithin uses a faster short-circuit distance function.

Examples

```
-- 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.geom, h.hospital_name
  FROM schools s
    LEFT JOIN hospitals h ON ST_DWithin(s.geom, h.geom, 3000)
  ORDER BY s.gid, ST_Distance(s.geom, h.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.geom, h.geom, 3000)
  WHERE h.gid IS NULL;

-- Find broadcasting towers that receiver with limited range can receive.
-- Data is geometry in Spherical Mercator (SRID=3857), ranges are approximate.

-- Create geometry index that will check proximity limit of user to tower
CREATE INDEX ON broadcasting_towers using gist (geom);

-- Create geometry index that will check proximity limit of tower to user
CREATE INDEX ON broadcasting_towers using gist (ST_Expand(geom, sending_range));

-- Query towers that 4-kilometer receiver in Minsk Hackerspace can get
-- Note: two conditions, because shorter LEAST(b.sending_range, 4000) will not use index.
SELECT b.tower_id, b.geom
  FROM broadcasting_towers b
  WHERE ST_DWithin(b.geom, 'SRID=3857;POINT(3072163.4 7159374.1)', 4000)
    AND ST_DWithin(b.geom, 'SRID=3857;POINT(3072163.4 7159374.1)', b.sending_range);
```

See Also

ST_Distance, ST_3DDWithin

7.11.2.5 ST_PointInsideCircle

ST_PointInsideCircle — Tests if a point geometry is inside a circle defined by a center and radius

Synopsis

boolean **ST_PointInsideCircle**(geometry a_point, float center_x, float center_y, float radius);

Description

Returns true if the geometry is a point and is inside the circle with center `center_x`,`center_y` and radius `radius`.

Warning
Does not use spatial indexes. Use ST_DWithin instead.

Availability: 1.2

Changed: 2.2.0 In prior versions this was called ST_Point_Inside_Circle

Examples

```
SELECT ST_PointInsideCircle(ST_Point(1,2), 0.5, 2, 3);
 st_pointinsidecircle
----------------------
 t
```

See Also

ST_DWithin

7.12 Measurement Functions

7.12.1 ST_Area

ST_Area — Retorna o centro geométrico de uma geometria.

Synopsis

float **ST_Area**(geometry g1);
float **ST_Area**(geography geog, boolean use_spheroid = true);

Descrição

Retorna a área da geometria se for um polígono ou multipolígono. Retorna a medida do comprimento de um valor ST_Surface ou ST_MultiSurface. Para geometria, uma área cartesiana 2D é determinada com unidades especificadas pelo SRID. Para geografia, por padrão, ela é determinada em um esferoide com unidade em metros quadrados. Para medir a esfera mais rápida, mas menos precisa, use: ST_Area(geog,false).

Melhorias: 2.0.0 - suporte a superfícies 2D poliédricas foi introduzido.

Melhorias: 2.2.0 - medição em esferoides desempenhada com GeographicLib para uma melhor precisão e força. Requer Proj >= 4.9.0 para tirar vantagem da nova característica.

Changed: 3.0.0 - does not depend on SFCGAL anymore.

This method implements the OGC Simple Features Implementation Specification for SQL 1.1.

This method implements the SQL/MM specification. SQL-MM 3: 8.1.2, 9.5.3

This function supports Polyhedral surfaces.

Note
Para superfícies poliédricas, somente suporta superfícies poliédricas 2D (não 2.5D). Para 2.5D, pode ser dada uma resposta não zero, mas somente para as faces que se encaixam completamente no plano XY.

Exemplos

Retorna uma área em pés quadrado para um terreno de Massachusetts e multiplica pela conversão para metros quadrados. Note que isto é em pés quadrados porque EPSG:2249 é o Massachusetts State Plane Feet

```
select ST_Area(geom) sqft,
    ST_Area(geom) * 0.3048 ^ 2 sqm
from (
         select 'SRID=2249;POLYGON((743238 2967416,743238 2967450,
                                  743265 2967450,743265.625 2967416,743238 2967416))' ::  ↩
                                     geometry geom
     ) subquery;
&#x250c;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x252c;&#x2500;&#x2
&#x2502;  sqft   &#x2502;      sqm      &#x2502;
&#x251c;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x253c;&#x2500;&#x2
&#x2502; 928.625 &#x2502; 86.27208552 &#x2502;
&#x2514;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2534;&#x2500;&#x2
```

Retorna uma área em pés quadrados e transforma para Massachusetts state plane em metros (EPSG:26986) para pegar metros quadrados. Note que ele é em pés quadrados, porque 2249 é Massachusetts State Plane Feet e a área transformada está em em metros quadrados já que EPSG:26986 é o state plane Massachusetts em metros

```
select ST_Area(geom) sqft,
    ST_Area(ST_Transform(geom, 26986)) As sqm
from (
         select
             'SRID=2249;POLYGON((743238 2967416,743238 2967450,
             743265 2967450,743265.625 2967416,743238 2967416))' :: geometry geom
     ) subquery;
&#x250c;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x252c;&#x2500;&#x2
&#x2502;  sqft   &#x2502;        sqm       &#x2502;
&#x251c;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x253c;&#x2500;&#x2
&#x2502; 928.625 &#x2502; 86.272430607008 &#x2502;
&#x2514;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2534;&#x2500;&#x2
```

Retorna uma área em pés quadrados e metros quadrados usado tipo de dados geografia. Note que transformamos nossa geometria para geografia (antes você pode certificar que sua geometria está em WGS 84 long lat 4326). A geografia sempre mede em metros. Isto é só para demonstração para comparar. Normalmente sua tabela já será armazenada no tipo de dados geografia.

```
select ST_Area(geog) / 0.3048 ^ 2 sqft_spheroid,
    ST_Area(geog, false) / 0.3048 ^ 2 sqft_sphere,
    ST_Area(geog) sqm_spheroid
from (
         select ST_Transform(
                    'SRID=2249;POLYGON((743238 2967416,743238 2967450,743265  ↩
                       2967450,743265.625 2967416,743238 2967416))'::geometry,
                    4326
```

```
            ) :: geography geog
    ) as subquery;
&#x250c;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2
&#x2502;  sqft_spheroid   &#x2502;   sqft_sphere    &#x2502;   sqm_spheroid   &#x2502;
&#x251c;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2
&#x2502; 928.684405784452 &#x2502; 927.049336105925 &#x2502; 86.2776044979692 &#x2502;
&#x2514;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2
```

If your data is in geography already:

```
select ST_Area(geog) / 0.3048 ^ 2 sqft,
    ST_Area(the_geog) sqm
from somegeogtable;
```

Veja também

ST_3DArea, ST_GeomFromEWKT, ST_LengthSpheroid, ST_Perimeter, ST_Transform

7.12.2 ST_Azimuth

ST_Azimuth — Retorna a menor linha 2-dimensional entre duas geometrias

Synopsis

float **ST_Azimuth**(geometry origin, geometry target);
float **ST_Azimuth**(geography origin, geography target);

Descrição

Returns the azimuth in radians of the target point from the origin point, or NULL if the two points are coincident. The azimuth angle is a positive clockwise angle referenced from the positive Y axis (geometry) or the North meridian (geography): North = 0; Northeast = $\pi/4$; East = $\pi/2$; Southeast = $3\pi/4$; South = π; Southwest $5\pi/4$; West = $3\pi/2$; Northwest = $7\pi/4$.

For the geography type, the azimuth solution is known as the inverse geodesic problem.

The azimuth is a mathematical concept defined as the angle between a reference vector and a point, with angular units in radians. The result value in radians can be converted to degrees using the PostgreSQL function `degrees()`.

Azimute é especialmente útil em conjunto com ST_Translate para mudar um objeto ao longo do seu eixo perpendicular. Veja upgis_lineshift Plpgsqlfunctions PostGIS wiki section para um exemplo disto.

Disponibilidade: 1.1.0

Melhorias: 2.0.0 suporte para geografia foi introduzido.

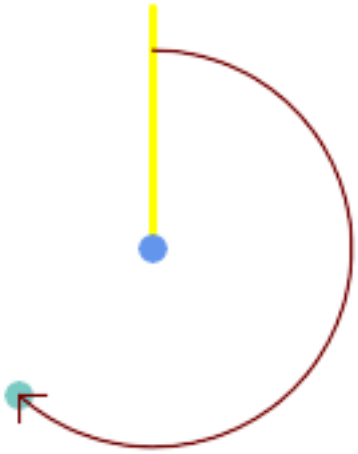
Melhorias: 2.2.0 medição em esferoides desempenhada com GeographicLib para uma melhor precisão e força. Requer Proj >= 4.9.0 para tirar vantagem da nova característica.

Exemplos

Azimute da geometria em graus

```
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
```

Blue: origin Point(25,45); Green: target Point(75, 100); Yellow: Y axis or North; Red: azimuth angle.



Blue: origin Point(75, 100); Green: target Point(25, 45); Yellow: Y axis or North; Red: azimuth angle.

Veja também

ST_Angle, ST_Translate, ST_Project, PostgreSQL Math Functions

7.12.3 ST_Angle

ST_Angle — Retorna a linha 3-dimensional mais longa entre duas geometrias

Synopsis

float **ST_Angle**(geometry point1, geometry point2, geometry point3, geometry point4);
float **ST_Angle**(geometry line1, geometry line2);

Descrição

Retorna a linha 3-dimensional mais longa entre duas geometrias

Variant 1: computes the angle enclosed by the points P1-P2-P3. If a 4th point provided computes the angle points P1-P2 and P3-P4

Variant 2: computes the angle between two vectors S1-E1 and S2-E2, defined by the start and end points of the input lines

O azimute é matematicamente conceituado como o ângulo entre um plano de referência e um ponto, com unidades angulares em radianos. As unidades podem ser convertidas para graus usando uma função PostgreSQL graus () embutida, como mostrado no exemplo.

Note that `ST_Angle(P1,P2,P3) = ST_Angle(P2,P1,P2,P3)`.

Availability: 2.5.0

Exemplos

linha mais longa entre polígono e polígono

```
SELECT degrees( ST_Angle('POINT(0 0)', 'POINT(10 10)', 'POINT(20 0)') );

 degrees
---------
     270
```

Angle between vectors defined by four points

```
SELECT degrees( ST_Angle('POINT (10 10)', 'POINT (0 0)', 'POINT(90 90)', 'POINT (100 80)')  ↩
    );

      degrees
--------------------
 269.9999999999999
```

Angle between vectors defined by the start and end points of lines

```
SELECT degrees( ST_Angle('LINESTRING(0 0, 0.3 0.7, 1 1)', 'LINESTRING(0 0, 0.2 0.5, 1 0)')  ↩
    );

      degrees
--------------
           45
```

Veja também

ST_Azimuth

7.12.4 ST_ClosestPoint

ST_ClosestPoint — Returns the 2D point on g1 that is closest to g2. This is the first point of the shortest line from one geometry to the other.

Synopsis

geometry **ST_ClosestPoint**(geometry geom1, geometry geom2);
geography **ST_ClosestPoint**(geography geom1, geography geom2, boolean use_spheroid = true);

Descrição

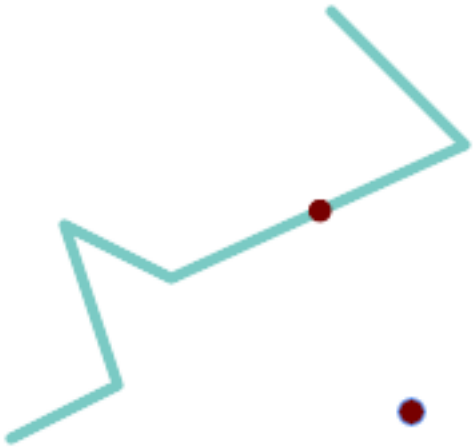
Returns the 2-dimensional point on `geom1` that is closest to `geom2`. This is the first point of the shortest line between the geometries (as computed by ST_ShortestLine).

Note
Se você tem uma geometria 3D, talvez prefira usar ST_3DClosestPoint.

Enhanced: 3.4.0 - Support for geography.

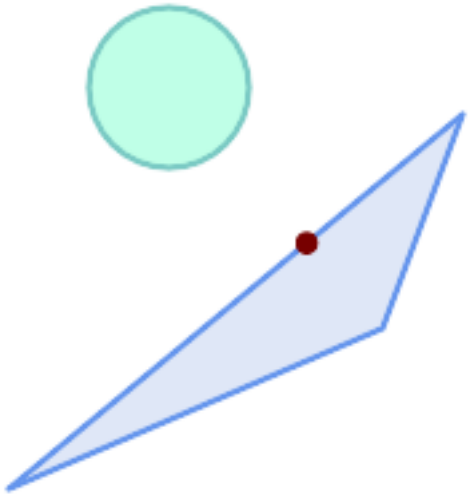
Disponibilidade: 1.5.0

Exemplos



The closest point for a Point and a LineString is the point itself. The closest point for a LineString and a Point is a point on the line.

```
SELECT ST_AsText( ST_ClosestPoint(pt,line)) AS cp_pt_line,
       ST_AsText( ST_ClosestPoint(line,pt)) AS cp_line_pt
    FROM (SELECT 'POINT (160 40)'::geometry AS pt,
                 'LINESTRING (10 30, 50 50, 30 110, 70 90, 180 140, 130 190)'::geometry AS  ↩
                    line ) AS t;

   cp_pt_line   |               cp_line_pt
----------------+-----------------------------------------
 POINT(160 40)  | POINT(125.75342465753425 115.34246575342466)
```



The closest point on polygon A to polygon B

```
SELECT ST_AsText( ST_ClosestPoint(
                'POLYGON ((190 150, 20 10, 160 70, 190 150))',
                ST_Buffer('POINT(80 160)', 30)        )) As ptwkt;
------------------------------------------
 POINT(131.59149149528952 101.89887534906197)
```

Veja também

ST_3DClosestPoint, ST_Distance, ST_LongestLine, ST_ShortestLine, ST_MaxDistance

7.12.5 ST_3DClosestPoint

ST_3DClosestPoint — Retorna o ponto 3 dimensional em g1 que é o mais próximo de g2. Este é o primeiro ponto da linha mais curta em três dimensões.

Synopsis

geometry **ST_3DClosestPoint**(geometry g1, geometry g2);

Descrição

Retorne o ponto 3-dimensional no g1 que é mais perto ao g2. Esse é o primeiro ponto da menor linha 3D. O comprimento 3D da menor linha 3D é a distância 3D.

 This function supports 3d and will not drop the z-index.

 This function supports Polyhedral surfaces.

Disponibilidade: 2.0.0

Alterações: 2.2.0 - se 2 geometrias 2D são entradas, um ponto 2D retorna (em vez do antigo comportamento assumindo 0 para Z perdido). Em caso de 2D e 3D, o Z não é mais 0 para Z perdido.

Exemplos

linestring e ponto -- pontos 3d e 2d mais próximos

```
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 e multiponto -- pontos 3d e 2d mais próximos

```
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 e polígono pontos 3d e 2d mais próximos

```
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)
```

Veja também

ST_AsEWKT, ST_ClosestPoint, ST_3DDistance, ST_3DShortestLine

7.12.6 ST_Distance

ST_Distance — Retorna a linha 3-dimensional mais longa entre duas geometrias

Synopsis

float **ST_Distance**(geometry g1, geometry g2);
float **ST_Distance**(geography geog1, geography geog2, boolean use_spheroid = true);

Descrição

Para tipo geometria, retorna a menor distância cartesiana 3-dimensional entre duas geometrias em unidades projetadas (spatial ref units).

For geografia types defaults to return the minimum geodesic distance between two geographies in meters, compute on the spheroid determined by the SRID. If `use_spheroid` is false, a faster spherical calculation is used.

 This method implements the OGC Simple Features Implementation Specification for SQL 1.1.

 This method implements the SQL/MM specification. SQL-MM 3: 5.1.23

 This method supports Circular Strings and Curves.

Disponibilidade: 1.5.0 suporte de geografia foi introduzido em 1.5. Melhorias na velocidade para planar para lidar melhor com mais ou maiores vértices de geometrias.

Melhorias: 2.1.0 velocidade melhorada para geografia. Veja Making Geography faster para mais detalhes.

Melhorias: 2.1.0 - suporte para geometrias curvas foi introduzido.

Melhorias: 2.2.0 - medição em esferoides desempenhada com GeographicLib para uma melhor precisão e força. Requer Proj >= 4.9.0 para tirar vantagem da nova característica.

Changed: 3.0.0 - does not depend on SFCGAL anymore.

Exemplos de Geometria

Geometry example - units in planar degrees 4326 is WGS 84 long lat, units are degrees.

```
SELECT ST_Distance(
    'SRID=4326;POINT(-72.1235 42.3521)'::geometry,
    'SRID=4326;LINESTRING(-72.1260 42.45, -72.123 42.1546)'::geometry );
-----------------
0.00150567726382282
```

Geometry example - units in meters (SRID: 3857, proportional to pixels on popular web maps). Although the value is off, nearby ones can be compared correctly, which makes it a good choice for algorithms like KNN or KMeans.

```
SELECT ST_Distance(
    ST_Transform('SRID=4326;POINT(-72.1235 42.3521)'::geometry, 3857),
    ST_Transform('SRID=4326;LINESTRING(-72.1260 42.45, -72.123 42.1546)'::geometry, 3857) ) ↩
        ;
-----------------
167.441410065196
```

Geometry example - units in meters (SRID: 3857 as above, but corrected by cos(lat) to account for distortion)

```
SELECT ST_Distance(
    ST_Transform('SRID=4326;POINT(-72.1235 42.3521)'::geometry, 3857),
    ST_Transform('SRID=4326;LINESTRING(-72.1260 42.45, -72.123 42.1546)'::geometry, 3857)
                ) * cosd(42.3521);
-----------------
123.742351254151
```

Geometry example - units in meters (SRID: 26986 Massachusetts state plane meters) (most accurate for Massachusetts)

```
SELECT ST_Distance(
    ST_Transform('SRID=4326;POINT(-72.1235 42.3521)'::geometry, 26986),
    ST_Transform('SRID=4326;LINESTRING(-72.1260 42.45, -72.123 42.1546)'::geometry, 26986)  ↩
        );
-----------------
123.797937878454
```

Geometry example - units in meters (SRID: 2163 US National Atlas Equal area) (least accurate)

```
SELECT ST_Distance(
    ST_Transform('SRID=4326;POINT(-72.1235 42.3521)'::geometry, 2163),
    ST_Transform('SRID=4326;LINESTRING(-72.1260 42.45, -72.123 42.1546)'::geometry, 2163) ) ↩
        ;
------------------
126.664256056812
```

Exemplos de Geografia

Same as geometry example but note units in meters - use sphere for slightly faster and less accurate computation.

```
SELECT ST_Distance(gg1, gg2) As spheroid_dist, ST_Distance(gg1, gg2, false) As sphere_dist
FROM (SELECT
    'SRID=4326;POINT(-72.1235 42.3521)'::geography as gg1,
    'SRID=4326;LINESTRING(-72.1260 42.45, -72.123 42.1546)'::geography as gg2
        ) As foo  ;

  spheroid_dist   |   sphere_dist
------------------+------------------
 123.802076746848 | 123.475736916397
```

Veja também

ST_3DDistance, ST_DWithin, ST_DistanceSphere, ST_DistanceSpheroid, ST_MaxDistance, ST_HausdorffDistance, ST_FrechetDistan
ST_Transform

7.12.7 ST_3DDistance

ST_3DDistance — Para tipo geometria, retorna a menor distância cartesiana 3-dimensional (baseado no sistema de referência espacial) entre duas geometrias em unidades projetadas.

Synopsis

float **ST_3DDistance**(geometry g1, geometry g2);

Descrição

Para tipo geometria, retorna a menor distância cartesiana 3-dimensional entre duas geometrias em unidades projetadas (spatial ref units).

 This function supports 3d and will not drop the z-index.

 This function supports Polyhedral surfaces.

 This method implements the SQL/MM specification. SQL-MM ISO/IEC 13249-3

Disponibilidade: 2.0.0

Alterações: 2.2.0 - Em caso de 2D e 3D, o Z não é mais 0 para Z perdido.

Changed: 3.0.0 - SFCGAL version removed

Exemplos

```
-- Geometry example - units in meters (SRID: 2163 US National Atlas Equal area) (3D point  ↩
    and line compared 2D point and line)
-- Note: currently no vertical datum support so Z is not transformed and assumed to be same ↩
     units as final.
SELECT ST_3DDistance(
                        ST_Transform('SRID=4326;POINT(-72.1235 42.3521 4)'::geometry,2163),
                        ST_Transform('SRID=4326;LINESTRING(-72.1260 42.45 15, -72.123  ↩
                            42.1546 20)'::geometry,2163)
                ) As dist_3d,
                ST_Distance(
                        ST_Transform('SRID=4326;POINT(-72.1235 42.3521)'::geometry,2163),
                        ST_Transform('SRID=4326;LINESTRING(-72.1260 42.45, -72.123 42.1546) ↩
                            '::geometry,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  'POLYGON((175 150 5, 20 40 5, 35 45 5, 50 60 5, 100 100 5, 175 150 5) ↩
            )'::geometry as poly,
                'MULTILINESTRING((175 155 2, 20 40 20, 50 60 -2, 125 100 1, 175 155 1), (1  ↩
                    10 2, 5 20 1))'::geometry as mline) as foo;
      dist3d       | dist2d
-------------------+--------
 0.716635696066337 |      0
```

Veja também

ST_Distance, ST_3DClosestPoint, ST_3DDWithin, ST_3DMaxDistance, ST_3DShortestLine, ST_Transform

7.12.8 ST_DistanceSphere

ST_DistanceSphere — Retorna a menor distância entre duas geometrias lon/lat dado um esferoide específico. As versões anteriores a 1.5 só suportam pontos.

Synopsis

float **ST_DistanceSphere**(geometry geomlonlatA, geometry geomlonlatB, float8 radius=6371008);

Descrição

Retorna a distância mínima em metros entre dois pontos long/lat. Usa uma terra esférica e raio derivado do esferoide definido pelo SRID. Mais rápido que ST_DistanceSpheroid, mas menos preciso. As versões do PostGIS anteriores a 1.5 só implementavam para pontos.

Disponibilidade: 1.5 - suporte para outros tipos de geometria além de pontos foi introduzido. As versões anteriores só funcionam com pontos.

Alterações: 2.2.0 Em versões anteriores era chamada de ST_Distance_Sphere

Exemplos

```
SELECT round(CAST(ST_DistanceSphere(ST_Centroid(geom), ST_GeomFromText('POINT(-118 38) ↩
    ',4326)) As numeric),2) As dist_meters,
round(CAST(ST_Distance(ST_Transform(ST_Centroid(geom),32611),
                ST_Transform(ST_GeomFromText('POINT(-118 38)', 4326),32611)) As numeric),2) ↩
                    As dist_utm11_meters,
round(CAST(ST_Distance(ST_Centroid(geom), ST_GeomFromText('POINT(-118 38)', 4326)) As  ↩
    numeric),5) As dist_degrees,
round(CAST(ST_Distance(ST_Transform(geom,32611),
                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 geom) ↩
            as foo;
         dist_meters | dist_utm11_meters | dist_degrees | min_dist_line_point_meters
        -------------+-------------------+--------------+----------------------------
                70424.47 |          70438.00 |      0.72900 |                   65871.18
```

Veja também

ST_Distance, ST_DistanceSpheroid

7.12.9 ST_DistanceSpheroid

ST_DistanceSpheroid — Retorna a menor distância entre duas geometrias lon/lat dado um esferoide específico. As versões anteriores a 1.5 só suportam pontos.

Synopsis

float **ST_DistanceSpheroid**(geometry geomlonlatA, geometry geomlonlatB, spheroid measurement_spheroid=WGS84);

Descrição

Retorna a distância mínima em metros entre duas geometrias long/lat dado um esferoide específico. Veja a explanação dos esferoides dados pela ST_LengthSpheroid. As versões do PostGIS anteriores a 1.5 só suportam pontos.

Note
Esta função não olha o SRID de uma geometria e sempre irá assumir que está representada nas coordenadas do esferoide passado. AS versões anteriores desta função só suportam pontos.

Disponibilidade: 1.5 - suporte para outros tipos de geometria além de pontos foi introduzido. As versões anteriores só funcionam com pontos.

Alterações: 2.2.0 Em versões anteriores era chamada de ST_Distance_Spheroid

Exemplos

```
SELECT round(CAST(
                ST_DistanceSpheroid(ST_Centroid(geom), ST_GeomFromText('POINT(-118 38) ↩
                    ',4326), 'SPHEROID["WGS 84",6378137,298.257223563]')
                        As numeric),2) As dist_meters_spheroid,
                round(CAST(ST_DistanceSphere(ST_Centroid(geom), ST_GeomFromText('POINT(-118 ↩
                     38)',4326)) As numeric),2) As dist_meters_sphere,
round(CAST(ST_Distance(ST_Transform(ST_Centroid(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 geom) ↩
            as foo;
 dist_meters_spheroid | dist_meters_sphere | dist_utm11_meters
----------------------+--------------------+-------------------
                       70454.92 |           70424.47 |          70438.00
```

Veja também

ST_Distance, ST_DistanceSphere

7.12.10 ST_FrechetDistance

ST_FrechetDistance — Retorna a menor linha 3-dimensional entre duas geometrias

Synopsis

float **ST_FrechetDistance**(geometry g1, geometry g2, float densifyFrac = -1);

Descrição

Implements algorithm for computing the Fréchet distance restricted to discrete points for both geometries, based on Computing Discrete Fréchet Distance. The Fréchet distance is a measure of similarity between curves that takes into account the location and ordering of the points along the curves. Therefore it is often better than the Hausdorff distance.

When the optional densifyFrac is specified, this function performs a segment densification before computing the discrete Fréchet distance. The densifyFrac parameter sets the fraction by which to densify each segment. Each segment will be split into a number of equal-length subsegments, whose fraction of the total length is closest to the given fraction.

Units are in the units of the spatial reference system of the geometries.

Note
A implementação atual suporta somente vértices como as localizações completas. Isto poderia ser expandido para permitir densidade arbitrária de pontos a serem usados.

Note
The smaller densifyFrac we specify, the more acurate Fréchet distance we get. But, the computation time and the memory usage increase with the square of the number of subsegments.

Desempenhado pelo módulo GEOS

Availability: 2.4.0 - requires GEOS >= 3.7.0

Exemplos

```
postgres=# SELECT st_frechetdistance('LINESTRING (0 0, 100 0)'::geometry, 'LINESTRING (0 0, ↩
     50 50, 100 0)'::geometry);
 st_frechetdistance
--------------------
   70.7106781186548
(1 row)
```

```
SELECT st_frechetdistance('LINESTRING (0 0, 100 0)'::geometry, 'LINESTRING (0 0, 50 50, 100 ↩
     0)'::geometry, 0.5);
 st_frechetdistance
--------------------
                 50
(1 row)
```

Veja também

ST_HausdorffDistance

7.12.11 ST_HausdorffDistance

ST_HausdorffDistance — Retorna a menor linha 3-dimensional entre duas geometrias

Synopsis

float **ST_HausdorffDistance**(geometry g1, geometry g2);
float **ST_HausdorffDistance**(geometry g1, geometry g2, float densifyFrac);

Descrição

Returns the Hausdorff distance between two geometries. The Hausdorff distance is a measure of how similar or dissimilar 2 geometries are.

The function actually computes the "Discrete Hausdorff Distance". This is the Hausdorff distance computed at discrete points on the geometries. The *densifyFrac* parameter can be specified, to provide a more accurate answer by densifying segments before computing the discrete Hausdorff distance. Each segment is split into a number of equal-length subsegments whose fraction of the segment length is closest to the given fraction.

Units are in the units of the spatial reference system of the geometries.

Note
This algorithm is NOT equivalent to the standard Hausdorff distance. However, it computes an approximation that is correct for a large subset of useful cases. One important case is Linestrings that are roughly parallel to each other, and roughly equal in length. This is a useful metric for line matching.

Disponibilidade: 1.5.0

Exemplos

Hausdorff distance (red) and distance (yellow) between two lines

```
SELECT ST_HausdorffDistance(geomA, geomB),
       ST_Distance(geomA, geomB)
    FROM (SELECT 'LINESTRING (20 70, 70 60, 110 70, 170 70)'::geometry AS geomA,
                 'LINESTRING (20 90, 130 90, 60 100, 190 100)'::geometry AS geomB) AS t;
 st_hausdorffdistance | st_distance
----------------------+-------------
    37.26206567625497 |          20
```

Example: Hausdorff distance with densification.

```
SELECT ST_HausdorffDistance(
            'LINESTRING (130 0, 0 0, 0 150)'::geometry,
            'LINESTRING (10 10, 10 150, 130 10)'::geometry,
            0.5);
 ----------------------
          70
```

Example: For each building, find the parcel that best represents it. First we require that the parcel intersect with the building geometry. `DISTINCT ON` guarantees we get each building listed only once. `ORDER BY .. ST_HausdorffDistance` selects the 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);
```

Veja também

ST_FrechetDistance

7.12.12 ST_Length

ST_Length — Retorna o centro geométrico de uma geometria.

Synopsis

float **ST_Length**(geometry a_2dlinestring);
float **ST_Length**(geography geog, boolean use_spheroid = true);

Descrição

Para geometria: Retorna o comprimento cartesiano 2D se for uma LineString, MultiLineString, ST_Curve, ST_MultiCurve. Retorna 0 para geometrias areais. Use ST_Perimeter. Para tipos de geometrias, unidades para medição de comprimento estão especificadas pelo sistema de referência espacial da geometria.

For geography types: computation is performed using the inverse geodesic calculation. Units of length are in meters. If PostGIS is compiled with PROJ version 4.8.0 or later, the spheroid is specified by the SRID, otherwise it is exclusive to WGS84. If `use_spheroid = false`, then the calculation is based on a sphere instead of a spheroid.

No momento, para geometria, isto é heterômio para ST_Length2D, mas isto pode mudar para dimensões maiores.

Warning
Alterações: 2.0.0 Quebrando a mudança -- nas versões anteriores aplicar isto a um MULTI/POLÍGONO de tipo de geografia lhe daria o perímetro do POLÍGONO/MULTIPOLÍGONO. Na 2.0.0 isso é alterado para retornar 0 a estar na linha com o comportamento da geometria. Por favor, utilize a ST_Perimeter se quiser o perímetro de um polígono

Note
Para a medição de geografia o padrão é a medição do esferoide. Para usar a esfera mais rápida e menos precisa, use ST_Length(gg,false);


This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.5.1


This method implements the SQL/MM specification. SQL-MM 3: 7.1.2, 9.3.4

Disponibilidade: 1.5.0 suporte para geografia foi introduzido em 1.5.

Exemplos de Geometria

Retorna o comprimento em pés para line string. Note que é em pés, porque EPSG:2249 é Massachusetts 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)'),
                26986
        )
);

st_length
---------
34309.4563576191
```

Exemplos de Geografia

Retorna o comprimento de WGS 84 linha de geografia

```
-- the default calculation uses a 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
------------------+------------------
 34310.5703627288 | 34346.2060960742
```

Veja também

ST_GeographyFromText, ST_GeomFromEWKT, ST_LengthSpheroid, ST_Perimeter, ST_Transform

7.12.13 ST_Length2D

ST_Length2D — Retorna o comprimento 2-dimensional da geometria se for uma linestring ou multi-linestring. Isto é um heterônimo para `ST_Length`

Synopsis

float **ST_Length2D**(geometry a_2dlinestring);

Descrição

Retorna o comprimento 2-dimensional da geometria se for uma linestring ou multi-linestring. Isto é um heterônimo para `ST_Length`

Veja também

ST_Length, ST_3DLength

7.12.14 ST_3DLength

ST_3DLength — Retorna o centro geométrico de uma geometria.

Synopsis

float **ST_3DLength**(geometry a_3dlinestring);

Descrição

Retorna o comprimento 3-dimensional ou 2-dimensional da geometria se for uma linestring ou multi-linestring. Para linhas 2-d, ela só retornará o comprimento 2-d (o mesmo da ST_Length e ST_Length2D)

 This function supports 3d and will not drop the z-index.

 This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 7.1, 10.3

Alterações: 2.0.0 Nas versões anteriores era chamado de ST_Length3D

Exemplos

Retorna o comprimento em pés para um cabo 3D. Note que é em pés, porque EPSG:2249 é Massachusetts 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
```

Veja também

ST_Length, ST_Length2D

7.12.15 ST_LengthSpheroid

ST_LengthSpheroid — Retorna o centro geométrico de uma geometria.

Synopsis

float **ST_LengthSpheroid**(geometry a_geometry, spheroid a_spheroid);

Descrição

Calcula o comprimento/perímetro de uma geometria em um elipsoide. É útil se as coordenadas da geometria estão em longitude/latitude e um comprimento é desejado sem reprojeção. O elipsoide é um tipo de banco de dados separado e pode ser construído como segue:

```
SPHEROID[<NAME>,<SEMI-MAJOR AXIS>,<INVERSE FLATTENING>]
```

Exemplo de geometria

```
SPHEROID["GRS_1980",6378137,298.257222101]
```

Disponibilidade: 1.2.2

Alterações: 2.2.0 Em versões anteriores era chamada de ST_Length_Spheroid e costumava ter um heterônimo ST_3DLength_Spheroid

 This function supports 3d and will not drop the z-index.

Exemplos

```
SELECT ST_LengthSpheroid( geometry_column,
                          'SPHEROID["GRS_1980",6378137,298.257222101]' )
                          FROM geometry_table;

SELECT ST_LengthSpheroid( geom, sph_m ) As tot_len,
ST_LengthSpheroid(ST_GeometryN(geom,1), sph_m) As len_line1,
ST_LengthSpheroid(ST_GeometryN(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 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_LengthSpheroid( geom, sph_m ) As tot_len,
ST_LengthSpheroid(ST_GeometryN(geom,1), sph_m) As len_line1,
ST_LengthSpheroid(ST_GeometryN(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 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
```

Veja também

ST_GeometryN, ST_Length

7.12.16 ST_LongestLine

ST_LongestLine — Retorna a linha 3-dimensional mais longa entre duas geometrias

Synopsis

geometry **ST_LongestLine**(geometry g1, geometry g2);

Descrição

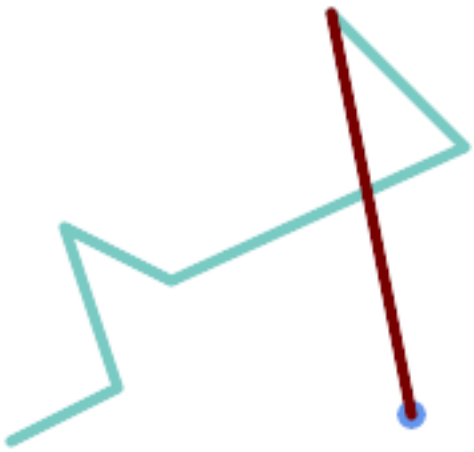
Returns the 2-dimensional longest line between the points of two geometries. The line returned starts on `g1` and ends on `g2`.

The longest line always occurs between two vertices. The function returns the first longest line if more than one is found. The length of the line is equal to the distance returned by ST_MaxDistance.

If g1 and g2 are the same geometry, returns the line between the two vertices farthest apart in the geometry. The endpoints of the line lie on the circle computed by ST_MinimumBoundingCircle.

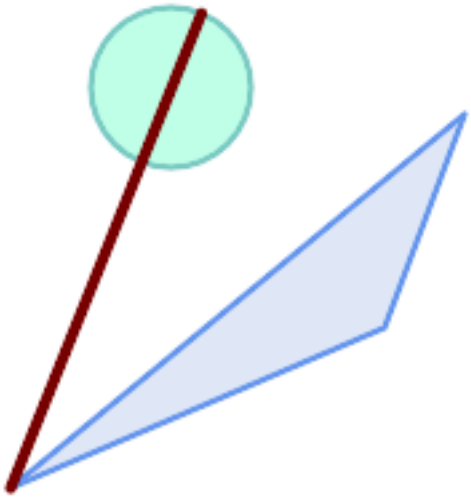
Disponibilidade: 1.5.0

Exemplos



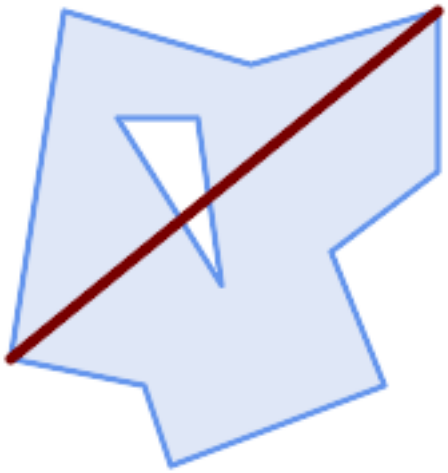
Linha mais longa entre ponto e linha

```
SELECT ST_AsText( ST_LongestLine(
        'POINT (160 40)',
        'LINESTRING (10 30, 50 50, 30 110, 70 90, 180 140, 130 190)' )
        ) AS lline;
-----------------
LINESTRING(160 40,130 190)
```



linha mais longa entre polígono e polígono

```
SELECT ST_AsText( ST_LongestLine(
        'POLYGON ((190 150, 20 10, 160 70, 190 150))',
        ST_Buffer('POINT(80 160)', 30)
            ) ) AS llinewkt;
-----------------
LINESTRING(20 10,105.3073372946034 186.95518130045156)
```



Longest line across a single geometry. The length of the line is equal to the Maximum Distance. The endpoints of the line lie on the Minimum Bounding Circle.

```
SELECT ST_AsText( ST_LongestLine( geom, geom)) AS llinewkt,
                  ST_MaxDistance( geom, geom) AS max_dist,
                  ST_Length( ST_LongestLine(geom, geom)) AS lenll
FROM (SELECT 'POLYGON ((40 180, 110 160, 180 180, 180 120, 140 90, 160 40, 80 10, 70 40, 20 ↩
     50, 40 180),
              (60 140, 99 77.5, 90 140, 60 140))'::geometry AS geom) AS t;

         llinewkt          |      max_dist      |       lenll
---------------------------+--------------------+--------------------
 LINESTRING(20 50,180 180) | 206.15528128088303 | 206.15528128088303
```

Veja também

ST_MaxDistance, ST_ShortestLine, ST_3DLongestLine, ST_MinimumBoundingCircle

7.12.17 ST_3DLongestLine

ST_3DLongestLine — Retorna a linha 3-dimensional mais longa entre duas geometrias

Synopsis

geometry **ST_3DLongestLine**(geometry g1, geometry g2);

Descrição

Retorna a linha 3-dimensional mais longa entre duas geometrias. A função só retornará a primeira linha, se existirem mais de uma. A linha retornada sempre começará em g1 e acabará em g2. O comprimento da linha essa função sempre será o mesmo do ST_3DMaxDistance retorna para g1 e g2.

Disponibilidade: 2.0.0

Alterações: 2.2.0 - se 2 geometrias 2D são entradas, um ponto 2D retorna (em vez do antigo comportamento assumindo 0 para Z perdido). Em caso de 2D e 3D, o Z não é mais 0 para Z perdido.

 This function supports 3d and will not drop the z-index.

 This function supports Polyhedral surfaces.

Exemplos

linestring e ponto -- linhas 3d e 2d mais longas

```
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 e multiponto -- linhas 3d e 2d mais longas

```
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 e polígono linhas 3d e 2d mais longas

```
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)
```

Veja também

ST_3DClosestPoint, ST_3DDistance, ST_LongestLine, ST_3DShortestLine, ST_3DMaxDistance

7.12.18 ST_MaxDistance

ST_MaxDistance — Retorna a maior distância 2-dimensional entre duas geometrias em unidades projetadas.

Synopsis

float **ST_MaxDistance**(geometry g1, geometry g2);

Descrição

Retorna a distância 2-dimensional máxima entre duas geometrias em unidades projetadas. Se g1 e g2 forem a mesma geometria, a função retornará a distância entre os dois vértices mais longes um do outro naquela geometria.

Retorna a distância 2-dimensional máxima entre duas geometrias em unidades projetadas. Se g1 e g2 forem a mesma geometria, a função retornará a distância entre os dois vértices mais longes um do outro naquela geometria.

Disponibilidade: 1.5.0

Exemplos

Linha mais longa entre ponto e linha

```
SELECT ST_MaxDistance('POINT(0 0)'::geometry, 'LINESTRING ( 2 0, 0 2 )'::geometry);
-----------------
 2

SELECT ST_MaxDistance('POINT(0 0)'::geometry, 'LINESTRING ( 2 2, 2 2 )'::geometry);
------------------
 2.82842712474619
```

Maximum distance between vertices of a single geometry.

```
SELECT ST_MaxDistance('POLYGON ((10 10, 10 0, 0 0, 10 10))'::geometry,
                      'POLYGON ((10 10, 10 0, 0 0, 10 10))'::geometry);
------------------
 14.142135623730951
```

Veja também

ST_Distance, ST_LongestLine, ST_DFullyWithin

7.12.19 ST_3DMaxDistance

ST_3DMaxDistance — Para tipo de geometria retorna a maior distância 3-dimensional cartesiana (baseada na referência espacial) entre duas geometrias em unidade projetadas.

Synopsis

float **ST_3DMaxDistance**(geometry g1, geometry g2);

Descrição

Para tipo geometria, retorna a menor distância cartesiana 3-dimensional entre duas geometrias em unidades projetadas (spatial ref units).

 This function supports 3d and will not drop the z-index.

 This function supports Polyhedral surfaces.

Disponibilidade: 2.0.0

Alterações: 2.2.0 - Em caso de 2D e 3D, o Z não é mais 0 para Z perdido.

Exemplos

```
-- Geometry example - units in meters (SRID: 2163 US National Atlas Equal area) (3D point  ↩
    and line compared 2D point and line)
-- Note: 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
```

Veja também

ST_Distance, ST_3DDWithin, ST_3DMaxDistance, ST_Transform

7.12.20 ST_MinimumClearance

ST_MinimumClearance — Retorna a liquidação mínima de uma geometria, uma medida de uma robustez de uma geometria.

Synopsis

float **ST_MinimumClearance**(geometry g);

Descrição

Não é incomum ter uma geometria que, enquanto o critério de validade de acordo com ST_IsValid (polígonos) ou ST_IsSimple (linhas), se tornaria inválida se um de seus vértices de movess por uma pequena distância, como pode acontecer durante uma conversão para formatos baseados em textos (como WKT, KML, GML GeoJSON), ou formatos binários que não usam coordenadas de pontos flutuantes de precisão dupla (MapInfo TAB).

The minimum clearance is a quantitative measure of a geometry's robustness to change in coordinate precision. It is the largest distance by which vertices of the geometry can be moved without creating an invalid geometry. Larger values of minimum clearance indicate greater robustness.

Se uma geometria tem uma liquidação mínima de `e`, pode ser dito que:

- Dois vértices distintos na geometria não são separados por menos que `e`.
- Nenhum vértice está mais perto que `e` a um segmento de linha do qual ele não é o endpoint.

Se nenhuma liquidação existe para uma geometria (por exemplo, um único ponto, ou um multiponto cujos pontos são idênticos), então a retornará infinita.

To avoid validity issues caused by precision loss, ST_ReducePrecision can reduce coordinate precision while ensuring that polygonal geometry remains valid.

Disponibilidade: 2.3.0

Exemplos

```
SELECT ST_MinimumClearance('POLYGON ((0 0, 1 0, 1 1, 0.5 3.2e-4, 0 0))');
 st_minimumclearance
---------------------
             0.00032
```

Veja também

ST_MinimumClearanceLine, ST_Crosses, ST_Dimension, ST_Intersects

7.12.21 ST_MinimumClearanceLine

ST_MinimumClearanceLine — Retorna a LineString de dois pontos abrangendo a liquidação mínima de uma geometria.

Synopsis

Geometry **ST_MinimumClearanceLine**(geometry g);

Descrição

Returns the two-point LineString spanning a geometry's minimum clearance. If the geometry does not have a minimum clearance, `LINESTRING EMPTY` is returned.

Desempenhado pelo módulo GEOS

Disponibilidade: 2.3.0 - requer GEOS >= 3.6.0

Exemplos

```
SELECT ST_AsText(ST_MinimumClearanceLine('POLYGON ((0 0, 1 0, 1 1, 0.5 3.2e-4, 0 0))'));
------------------------------
LINESTRING(0.5 0.00032,0.5 0)
```

Veja também

ST_MinimumClearance

7.12.22 ST_Perimeter

ST_Perimeter — Returns the length of the boundary of a polygonal geometry or geography.

Synopsis

float **ST_Perimeter**(geometry g1);
float **ST_Perimeter**(geography geog, boolean use_spheroid = true);

Descrição

Retorna o perímetro 2D da geometria/geografia se for uma ST_Surface, ST_MultiSurface (Polygon, MultiPolygon). Retorna 0 para geometrias não areais. Para geometrias lineares, use ST_Length. Para tipos de geometria, unidades para medição de perímetro estão especificadas pelo sistema de referência espacial da geometria.

For geography types, the calculations are performed using the inverse geodesic problem, where perimeter units are in meters. If PostGIS is compiled with PROJ version 4.8.0 or later, the spheroid is specified by the SRID, otherwise it is exclusive to WGS84. If `use_spheroid = false`, then calculations will approximate a sphere instead of a spheroid.

No momento isto é um heterônimo para ST_Perimeter2D, mas pode ser alterado para suportar dimensões maiores.

 This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.5.1

 This method implements the SQL/MM specification. SQL-MM 3: 8.1.3, 9.5.4

Disponibilidade 2.0.0: Suporte para geografia foi introduzido

Exemplos: Geometria

Retorna o perímetro em pés para Polígono e Multipolígono. Note que é em pés, porque EPSG:2249 é Massachusetts 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)
```

Exemplos: Geografia

Retorna perímetro em metros e pés para Polígono e MultiPolígono. Note que isso é geografia (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
```

Veja também

ST_GeogFromText, ST_GeomFromText, ST_Length

7.12.23 ST_Perimeter2D

ST_Perimeter2D — Returns the 2D perimeter of a polygonal geometry. Alias for `ST_Perimeter`.

Synopsis

float **ST_Perimeter2D**(geometry geomA);

Descrição

Retorna o perímetro 2-dimensional da geometria, se for uma polígono ou multi-polígono.

Note
Isto é um heterônimo para ST_Perimeter. Nas próximas versões a ST_Perimeter. pode retornar a maior dimensão de perímetro para uma geometria. Continua abaixo de consideração

Veja também

ST_Perimeter

7.12.24 ST_3DPerímetro

ST_3DPerímetro — Retorna o centro geométrico de uma geometria.

Synopsis

float **ST_3DPerimeter**(geometry geomA);

Descrição

Retorna o perímetro 3-dimensional da geometria, se for um polígono ou multi-polígono. Se a geometria for 2-dimensional, então retorna o perímetro 2-dimensional.


This function supports 3d and will not drop the z-index.


This method implements the SQL/MM specification. SQL-MM ISO/IEC 13249-3: 8.1, 10.5

Alterações: 2.0.0 Nas versões anteriores era chamado de ST_Perimeter3D

Exemplos

O perímetro de um polígono levemente elevado no ar no Massachusetts state plane feet

```
SELECT ST_3DPerimeter(geom), ST_Perimeter2d(geom), ST_Perimeter(geom) FROM
                        (SELECT ST_GeomFromEWKT('SRID=2249;POLYGON((743238 2967416 2,743238 ↩
                            2967450 1,
743265.625 2967416 1,743238 2967416 2))') As geom) As foo;

  ST_3DPerimeter   |  st_perimeter2d  |   st_perimeter
-------------------+------------------+------------------
 105.465793597674 | 105.432997272188 | 105.432997272188
```

Veja também

ST_GeomFromEWKT, ST_Perimeter, ST_Perimeter2D

7.12.25 ST_ShortestLine

ST_ShortestLine — Retorna a menor linha 2-dimensional entre duas geometrias

Synopsis

geometry **ST_ShortestLine**(geometry geom1, geometry geom2);
geography **ST_ShortestLine**(geography geom1, geography geom2, boolean use_spheroid = true);

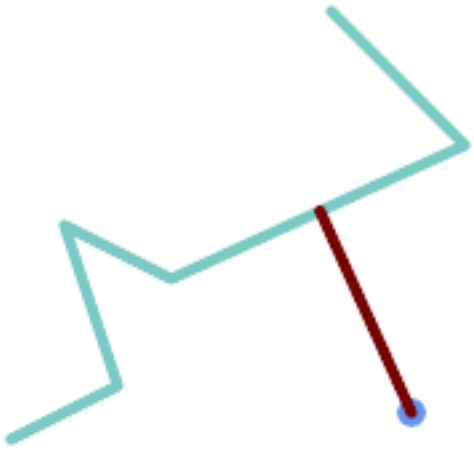
Descrição

Returns the 2-dimensional shortest line between two geometries. The line returned starts in `geom1` and ends in `geom2`. If `geom1` and `geom2` intersect the result is a line with start and end at an intersection point. The length of the line is the same as ST_Distance returns for g1 and g2.

Enhanced: 3.4.0 - support for geography.

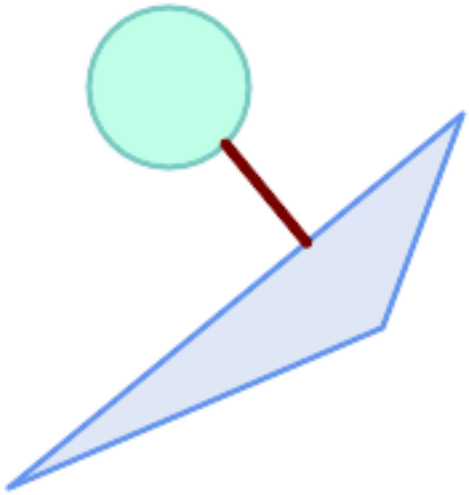
Disponibilidade: 1.5.0

Exemplos



Shortest line between Point and LineString

```
SELECT ST_AsText(  ST_ShortestLine(
        'POINT (160 40)',
        'LINESTRING (10 30, 50 50, 30 110, 70 90, 180 140, 130 190)')
        ) As sline;
-----------------------------------------------------
 LINESTRING(160 40,125.75342465753425 115.34246575342466)
```



Shortest line between Polygons

```
SELECT ST_AsText( ST_ShortestLine(
         'POLYGON ((190 150, 20 10, 160 70, 190 150))',
         ST_Buffer('POINT(80 160)', 30)
              ) ) AS llinewkt;
-----------------
LINESTRING(131.59149149528952 101.89887534906197,101.21320343559644 138.78679656440357)
```

Veja também

ST_ClosestPoint, ST_Distance, ST_LongestLine, ST_MaxDistance

7.12.26 ST_3DShortestLine

ST_3DShortestLine — Retorna a menor linha 3-dimensional entre duas geometrias

Synopsis

geometry **ST_3DShortestLine**(geometry g1, geometry g2);

Descrição

Retorna a menor linha 3-dimensional entre duas geometrias, a função só irá retornar a primeira linha menor se houverem mais de um, este a função encontra. Se g1 e g2 intersecta em apenas um ponto, a função retornará uma linha com os pontos de interseção da direita e esquerda. Se g1 e g2 estão intersectando em mais de um ponto, a função retornará uma linha com começo e fim no mesmo ponto, mas também pode ser qualquer um dos outros pontos. A linha que retorna sempre começará com g2 e acabará em g2. O comprimento 3D da linha, os retornos desta função serão sempre os mesmos dos retornos para g1 e g2 ST_3DDistance.

Disponibilidade: 2.0.0

Alterações: 2.2.0 - se 2 geometrias 2D são entradas, um ponto 2D retorna (em vez do antigo comportamento assumindo 0 para Z perdido). Em caso de 2D e 3D, o Z não é mais 0 para Z perdido.

This function supports 3d and will not drop the z-index.

This function supports Polyhedral surfaces.

Exemplos

linestring e ponto -- linhas 3d e 2d mais curtas

```
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 e multiponto -- linhas 3d e 2d mais curtas

```
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;


                        shl3d_line_pt                                   |  ↩
    shl2d_line_pt
------------------------------------------------------------------------------+----------------------

 LINESTRING(54.6993798867619 128.935022917228 11.5475869506606,100 100 30) | LINESTRING ↩
    (50 75,50 74)
```

Multilinestring e polígono linhas 3d e 2d mais curtas

```
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)
```

Veja também

ST_3DClosestPoint, ST_3DDistance, ST_LongestLine, ST_ShortestLine, ST_3DMaxDistance

7.13 Overlay Functions

7.13.1 ST_ClipByBox2D

ST_ClipByBox2D — Computes the portion of a geometry falling within a rectangle.

Synopsis

geometry **ST_ClipByBox2D**(geometry geom, box2d box);

Description

Clips a geometry by a 2D box in a fast and tolerant but possibly invalid way. Topologically invalid input geometries do not result in exceptions being thrown. The output geometry is not guaranteed to be valid (in particular, self-intersections for a polygon may be introduced).

Performed by the GEOS module.

Availability: 2.2.0

Examples

```
-- Rely on implicit cast from geometry to box2d for the second parameter
SELECT ST_ClipByBox2D(geom, ST_MakeEnvelope(0,0,10,10)) FROM mytab;
```

See Also

ST_Intersection, ST_MakeBox2D, ST_MakeEnvelope

7.13.2 ST_Difference

ST_Difference — Computes a geometry representing the part of geometry A that does not intersect geometry B.

Synopsis

geometry **ST_Difference**(geometry geomA, geometry geomB, float8 gridSize = -1);

Description

Returns a geometry representing the part of geometry A that does not intersect geometry B. This is equivalent to `A - ST_Intersecti`
If A is completely contained in B then an empty atomic geometry of appropriate type is returned.

Note
This is the only overlay function where input order matters. ST_Difference(A, B) always returns a portion of A.

If the optional `gridSize` argument is provided, the inputs are snapped to a grid of the given size, and the result vertices are computed on that same grid. (Requires GEOS-3.9.0 or higher)

Performed by the GEOS module

Enhanced: 3.1.0 accept a gridSize parameter.

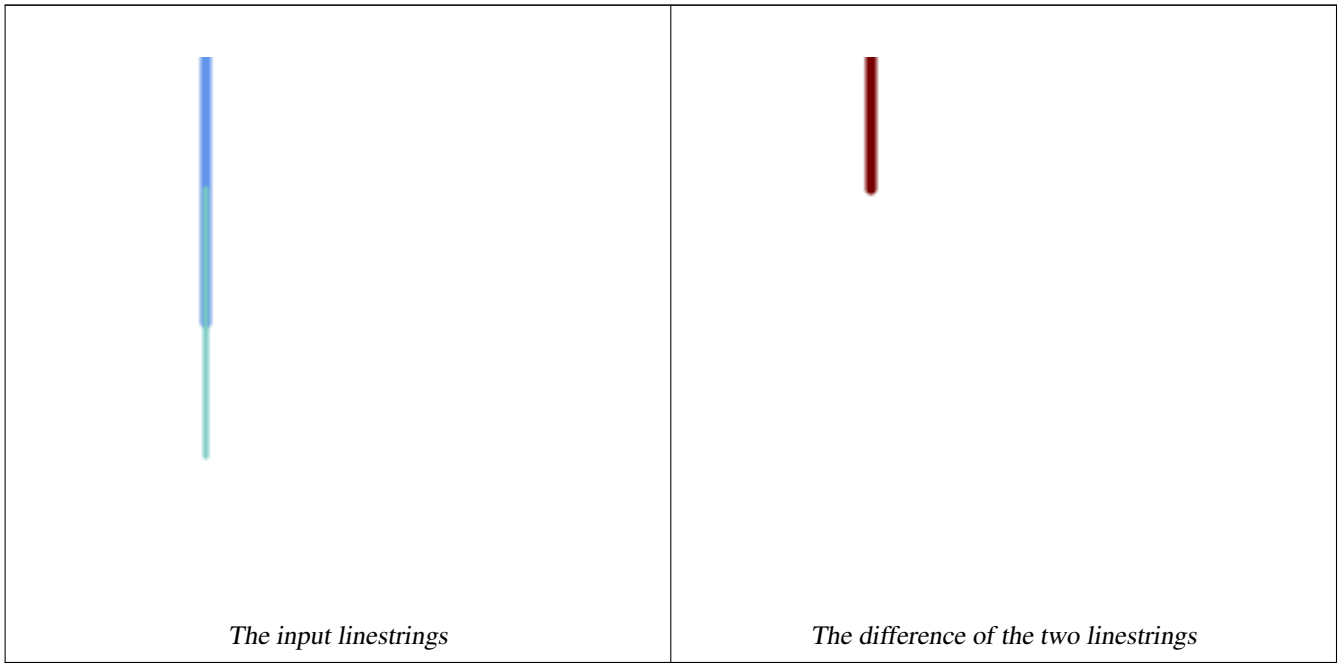
Requires GEOS >= 3.9.0 to use the gridSize parameter.

 This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.3

 This method implements the SQL/MM specification. SQL-MM 3: 5.1.20

 This function supports 3d and will not drop the z-index. However, the result is computed using XY only. The result Z values are copied, averaged or interpolated.

Examples



The input linestrings | *The difference of the two linestrings*

The difference of 2D linestrings.

```
SELECT ST_AsText(
    ST_Difference(
            'LINESTRING(50 100, 50 200)'::geometry,
            'LINESTRING(50 50, 50 150)'::geometry
        )
    );

st_astext
---------
LINESTRING(50 150,50 200)
```

The difference of 3D points.

```
SELECT ST_AsEWKT( ST_Difference(
                   'MULTIPOINT(-118.58 38.38 5,-118.60 38.329 6,-118.614 38.281 7)' ::  ↩
                       geometry,
                   'POINT(-118.614 38.281 5)' :: geometry
                  ) );

st_asewkt
---------
MULTIPOINT(-118.6 38.329 6,-118.58 38.38 5)
```

See Also

ST_SymDifference, ST_Intersection, ST_Union

7.13.3 ST_Intersection

ST_Intersection — Computes a geometry representing the shared portion of geometries A and B.

Synopsis

geometry **ST_Intersection**(geometry geomA , geometry geomB , float8 gridSize = -1);
geography **ST_Intersection**(geography geogA , geography geogB);

Description

Returns a geometry representing the point-set intersection of two geometries. In other words, that portion of geometry A and geometry B that is shared between the two geometries.

If the geometries have no points in common (i.e. are disjoint) then an empty atomic geometry of appropriate type is returned.

If the optional `gridSize` argument is provided, the inputs are snapped to a grid of the given size, and the result vertices are computed on that same grid. (Requires GEOS-3.9.0 or higher)

ST_Intersection in conjunction with ST_Intersects is useful for clipping geometries such as in bounding box, buffer, or region queries where you only require the portion of a geometry that is inside a country or region of interest.

Note

For geography this is a thin wrapper around the geometry implementation. It first determines the best SRID that fits the bounding box of the 2 geography objects (if geography objects are within one half zone UTM but not same UTM will pick one of those) (favoring UTM or Lambert Azimuthal Equal Area (LAEA) north/south pole, and falling back on mercator in worst case scenario) and then intersection in that best fit planar spatial ref and retransforms back to WGS84 geography.

Warning

This function will drop the M coordinate values if present.

Warning

If working with 3D geometries, you may want to use SFGCAL based ST_3DIntersection which does a proper 3D intersection for 3D geometries. Although this function works with Z-coordinate, it does an averaging of Z-Coordinate.

Performed by the GEOS module

Enhanced: 3.1.0 accept a gridSize parameter

Requires GEOS >= 3.9.0 to use the gridSize parameter

Changed: 3.0.0 does not depend on SFCGAL.

Availability: 1.5 support for geography data type was introduced.


This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.3


This method implements the SQL/MM specification. SQL-MM 3: 5.1.18


This function supports 3d and will not drop the z-index. However, the result is computed using XY only. The result Z values are copied, averaged or interpolated.

Examples

```
SELECT ST_AsText(ST_Intersection('POINT(0 0)'::geometry, 'LINESTRING ( 2 0, 0 2 )':: ↩
    geometry));
 st_astext
---------------
GEOMETRYCOLLECTION EMPTY

SELECT ST_AsText(ST_Intersection('POINT(0 0)'::geometry, 'LINESTRING ( 0 0, 0 2 )':: ↩
    geometry));
 st_astext
---------------
POINT(0 0)
```

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.geom, trails.geom))).geom clipped_geom
         from country
              inner join trails on ST_Intersects(country.geom, trails.geom)
     ) as clipped
where ST_Dimension(clipped.clipped_geom) = 1;
```

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.geom, poly.geom),
            0.0
        )
    ) clipped_geom
from country
     inner join poly on ST_Intersects(country.geom, poly.geom)
where not ST_IsEmpty(ST_Buffer(ST_Intersection(country.geom, poly.geom), 0.0));
```

Examples: 2.5Dish

Note this is not a true intersection, compare to the same example using ST_3DIntersection.

```
select ST_AsText(ST_Intersection(linestring, polygon)) As wkt
from  ST_GeomFromText('LINESTRING Z (2 2 6,1.5 1.5 7,1 1 8,0.5 0.5 8,0 0 10)') AS  ↩
    linestring
 CROSS JOIN ST_GeomFromText('POLYGON((0 0 8, 0 1 8, 1 1 8, 1 0 8, 0 0 8))') AS polygon;

               st_astext
-----------------------------------------
 LINESTRING Z (1 1 8,0.5 0.5 8,0 0 10)
```

See Also

ST_3DIntersection, ST_Difference, ST_Union, ST_Dimension, ST_Dump, ST_Force2D, ST_SymDifference, ST_Intersects, ST_Multi

7.13.4 ST_MemUnion

ST_MemUnion — Aggregate function which unions geometries in a memory-efficent but slower way

Synopsis

geometry **ST_MemUnion**(geometry set geomfield);

Description

An aggregate function that unions the input geometries, merging them to produce a result geometry with no overlaps. The output may be a single geometry, a MultiGeometry, or a Geometry Collection.

Note
Produces the same result as ST_Union, but uses less memory and more processor time. This aggregate function works by unioning the geometries incrementally, as opposed to the ST_Union aggregate which first accumulates an array and then unions the contents using a fast algorithm.

This function supports 3d and will not drop the z-index. However, the result is computed using XY only. The result Z values are copied, averaged or interpolated.

Examples

```
SELECT id,
       ST_MemUnion(geom) as singlegeom
FROM sometable f
GROU BY id;
```

See Also

ST_Union

7.13.5 ST_Node

ST_Node — Nodes a collection of lines.

Synopsis

geometry **ST_Node**(geometry geom);

Description

Returns a (Multi)LineString representing the fully noded version of a collection of linestrings. The noding preserves all of the input nodes, and introduces the least possible number of new nodes. The resulting linework is dissolved (duplicate lines are removed).

This is a good way to create fully-noded linework suitable for use as input to ST_Polygonize.

ST_UnaryUnion can also be used to node and dissolve linework. It provides an option to specify a gridSize, which can provide simpler and more robust output. See also ST_Union for an aggregate variant.

 This function supports 3d and will not drop the z-index.

Performed by the GEOS module.

Availability: 2.0.0

Changed: 2.4.0 this function uses GEOSNode internally instead of GEOSUnaryUnion. This may cause the resulting linestrings to have a different order and direction compared to PostGIS < 2.4.

Examples

Noding a 3D LineString which self-intersects

```
SELECT ST_AsText(
        ST_Node('LINESTRINGZ(0 0 0, 10 10 10, 0 10 5, 10 0 3)'::geometry)
    ) As  output;
output
-----------
MULTILINESTRING Z ((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))
```

Noding two LineStrings which share common linework. Note that the result linework is dissolved.

```
SELECT ST_AsText(
        ST_Node('MULTILINESTRING ((2 5, 2 1, 7 1), (6 1, 4 1, 2 3, 2 5))'::geometry)
    ) As  output;
output
-----------
MULTILINESTRING((2 5,2 3),(2 3,2 1,4 1),(4 1,2 3),(4 1,6 1),(6 1,7 1))
```

See Also

ST_UnaryUnion, ST_Union

7.13.6 ST_Split

ST_Split — Returns a collection of geometries created by splitting a geometry by another geometry.

Synopsis

geometry **ST_Split**(geometry input, geometry blade);

Description

The function supports splitting a LineString by a (Multi)Point, (Multi)LineString or (Multi)Polygon boundary, or a (Multi)Polygon by a LineString. When a (Multi)Polygon is used as as the blade, its linear components (the boundary) are used for splitting the input. The result geometry is always a collection.

This function is in a sense the opposite of ST_Union. Applying ST_Union to the returned collection should theoretically yield the original geometry (although due to numerical rounding this may not be exactly the case).

Note
If the the input and blade do not intersect due to numerical precision issues, the input may not be split as expected. To avoid this situation it may be necessary to snap the input to the blade first, using ST_Snap with a small tolerance.

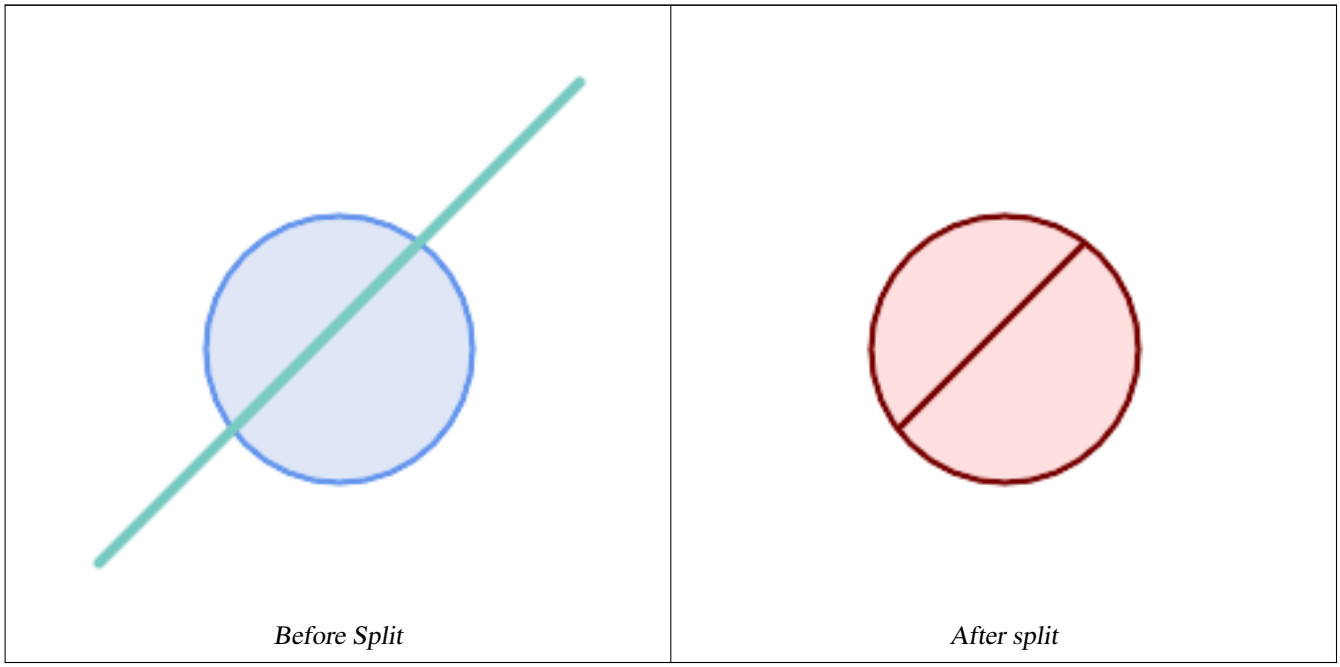
Availability: 2.0.0 requires GEOS

Enhanced: 2.2.0 support for splitting a line by a multiline, a multipoint or (multi)polygon boundary was introduced.

Enhanced: 2.5.0 support for splitting a polygon by a multiline was introduced.

Examples

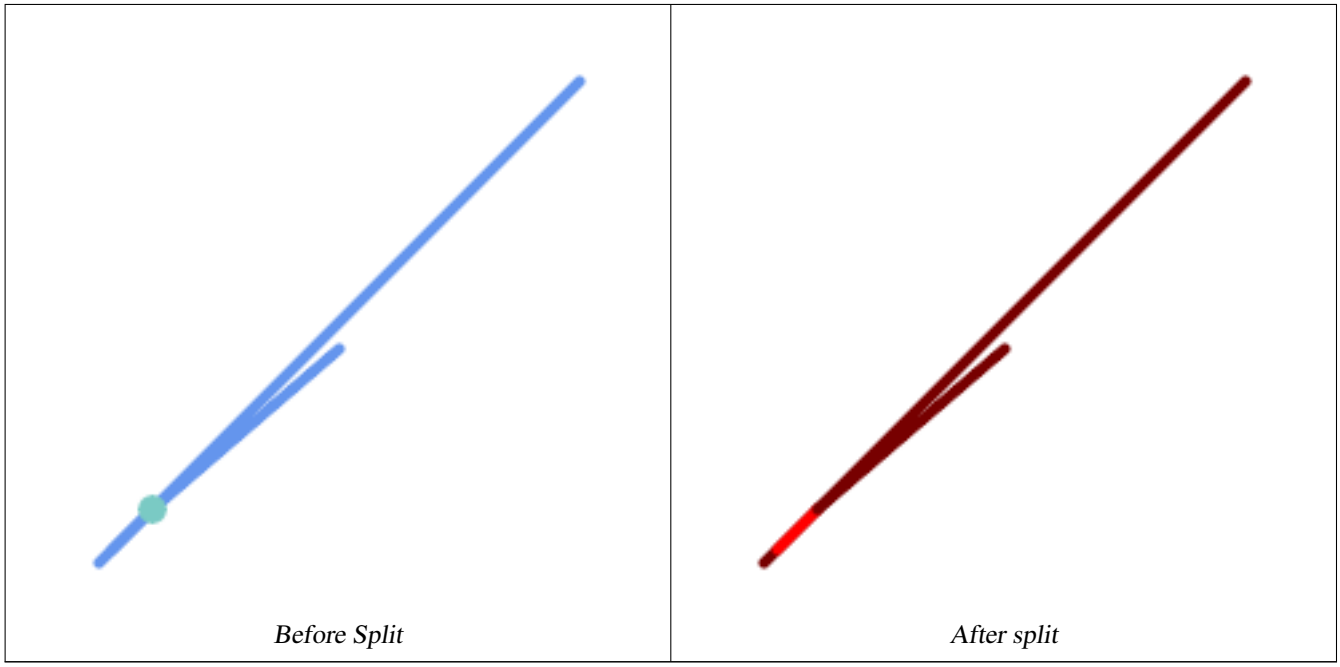
Split a Polygon by a Line.



Before Split | *After split*

```
SELECT ST_AsText( ST_Split(
                ST_Buffer(ST_GeomFromText('POINT(100 90)'), 50), -- circle
                ST_MakeLine(ST_Point(10, 10),ST_Point(190, 190)) -- line
    ));

-- result --
 GEOMETRYCOLLECTION(
            POLYGON((150 90,149.039264020162 80.2454838991936,146.193976625564  ↩
                70.8658283817455,..),
            POLYGON(..))
)
```

Split a MultiLineString a Point, where the point lies exactly on both LineStrings elements.



Before Split | *After split*

```
SELECT ST_AsText(ST_Split(
    'MULTILINESTRING((10 10, 190 191), (15 15, 30 30, 100 90))',
    ST_Point(30,30))) As split;

split
------
GEOMETRYCOLLECTION(
    LINESTRING(10 10,30 30),
    LINESTRING(30 30,190 190),
    LINESTRING(15 15,30 30),
    LINESTRING(30 30,100 90)
)
```

Split a LineString by a Point, where the point does not lie exactly on the line. Shows using ST_Snap to snap the line to the point to allow it to be split.

```
WITH data AS (SELECT
  'LINESTRING(0 0, 100 100)'::geometry AS line,
  'POINT(51 50)':: geometry AS point
)
SELECT ST_AsText( ST_Split( ST_Snap(line, point, 1), point)) AS snapped_split,
       ST_AsText( ST_Split(line, point)) AS not_snapped_not_split
       FROM data;

                            snapped_split                                |             ↩
                                not_snapped_not_split
-------------------------------------------------------------------------+-------------------------------

 GEOMETRYCOLLECTION(LINESTRING(0 0,51 50),LINESTRING(51 50,100 100)) | GEOMETRYCOLLECTION( ↩
     LINESTRING(0 0,100 100))
```

See Also

ST_Snap, ST_Union

7.13.7 ST_Subdivide

ST_Subdivide — Computes a rectilinear subdivision of a geometry.

Synopsis

setof geometry **ST_Subdivide**(geometry geom, integer max_vertices=256, float8 gridSize = -1);

Description

Returns a set of geometries that are the result of dividing `geom` into parts using rectilinear lines, with each part containing no more than `max_vertices`.

`max_vertices` must be 5 or more, as 5 points are needed to represent a closed box. `gridSize` can be specified to have clipping work in fixed-precision space (requires GEOS-3.9.0+).

Point-in-polygon and other spatial operations are normally faster for indexed subdivided datasets. Since the bounding boxes for the parts usually cover a smaller area than the original geometry bbox, index queries produce fewer "hit" cases. The "hit" cases are faster because the spatial operations executed by the index recheck process fewer points.

Note
This is a set-returning function (SRF) that return a set of rows containing single geometry values. It can be used in a SELECT list or a FROM clause to produce a result set with one record for each result geometry.

Performed by the GEOS module.

Availability: 2.2.0

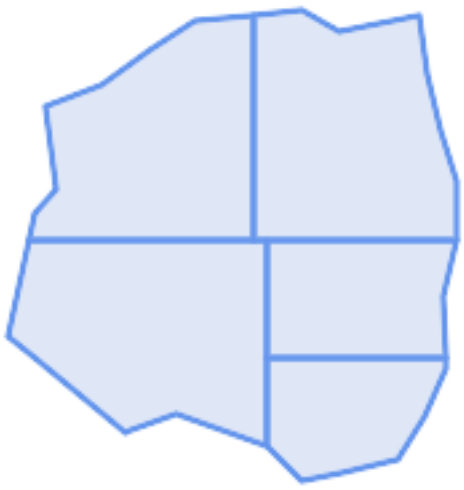
Enhanced: 2.5.0 reuses existing points on polygon split, vertex count is lowered from 8 to 5.

Enhanced: 3.1.0 accept a gridSize parameter.

Requires GEOS >= 3.9.0 to use the gridSize parameter

Examples

Example: Subdivide a polygon into parts with no more than 10 vertices, and assign each part a unique id.

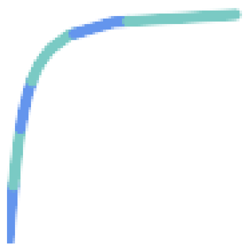


Subdivided to maximum 10 vertices

```
SELECT row_number() OVER() As rn, ST_AsText(geom) As wkt
    FROM (SELECT ST_SubDivide(
        'POLYGON((132 10,119 23,85 35,68 29,66 28,49 42,32 56,22 64,32 110,40 119,36 150,
        57 158,75 171,92 182,114 184,132 186,146 178,176 184,179 162,184 141,190 122,
        190 100,185 79,186 56,186 52,178 34,168 18,147 13,132 10))'::geometry,10))  AS f( ↩
            geom);
```

```
rn &#x2502;                                                      wkt
&#x2500;&#x2500;&#x2500;&#x2500;&#x253c;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2
  1 &#x2502; POLYGON((119 23,85 35,68 29,66 28,32 56,22 64,29.8260869565217 100,119 100,119 ↩
      23))
  2 &#x2502; POLYGON((132 10,119 23,119 56,186 56,186 52,178 34,168 18,147 13,132 10))
  3 &#x2502; POLYGON((119 56,119 100,190 100,185 79,186 56,119 56))
  4 &#x2502; POLYGON((29.8260869565217 100,32 110,40 119,36 150,57 158,75 171,92 182,114  ↩
     184,114 100,29.8260869565217 100))
  5 &#x2502; POLYGON((114 184,132 186,146 178,176 184,179 162,184 141,190 122,190 100,114  ↩
     100,114 184))
```

Example: Densify a long geography line using ST_Segmentize(geography, distance), and use ST_Subdivide to split the resulting line into sublines of 8 vertices.



The densified and split lines.

```
SELECT ST_AsText( ST_Subdivide(
            ST_Segmentize('LINESTRING(0 0, 85 85)'::geography,
                          1200000)::geometry,    8));
```

```
LINESTRING(0 0,0.487578359029357 5.57659056746196,0.984542144675897  ↩
    11.1527721155093,1.50101059639722 16.7281035483571,1.94532113630331 21.25)
LINESTRING(1.94532113630331 21.25,2.04869538062779 22.3020741387339,2.64204641967673  ↩
    27.8740533545155,3.29994062412787 33.443216802941,4.04836719489742  ↩
    39.0084282520239,4.59890468420694 42.5)
LINESTRING(4.59890468420694 42.5,4.92498503922732 44.5680389206321,5.98737409390639  ↩
    50.1195229244701,7.3290919767674 55.6587646879025,8.79638749938413 60.1969505994924)
LINESTRING(8.79638749938413 60.1969505994924,9.11375579533779  ↩
    61.1785363177625,11.6558166691368 66.6648504160202,15.642041247655  ↩
    72.0867690601745,22.8716627200212 77.3609628116894,24.6991785131552 77.8939011989848)
LINESTRING(24.6991785131552 77.8939011989848,39.4046096622744  ↩
    82.1822848017636,44.7994523421035 82.5156766227011)
LINESTRING(44.7994523421035 82.5156766227011,85 85)
```

Example: Subdivide the complex geometries of a table in-place. The original geometry records are deleted from the source table, and new records for each subdivided result geometry are inserted.

```
WITH complex_areas_to_subdivide AS (
    DELETE from polygons_table
    WHERE ST_NPoints(geom)
> 255
    RETURNING id, column1, column2, column3, geom
)
INSERT INTO polygons_table (fid, column1, column2, column3, geom)
    SELECT fid, column1, column2, column3,
           ST_Subdivide(geom, 255) as geom
    FROM complex_areas_to_subdivide;
```

Example: Create a new table containing subdivided geometries, retaining the key of the original geometry so that the new table can be joined to the source table. Since ST_Subdivide is a set-returning (table) function that returns a set of single-value rows, this syntax automatically produces a table with one row for each result part.

```
CREATE TABLE subdivided_geoms AS
    SELECT pkey, ST_Subdivide(geom) AS geom
    FROM original_geoms;
```

See Also

ST_ClipByBox2D, ST_Segmentize, ST_Split, ST_NPoints

7.13.8 ST_SymDifference

ST_SymDifference — Computes a geometry representing the portions of geometries A and B that do not intersect.

Synopsis

geometry **ST_SymDifference**(geometry geomA, geometry geomB, float8 gridSize = -1);

Description

Returns a geometry representing the portions of geonetries A and B that do not intersect. This is equivalent to `ST_Union(A,B) - ST_Intersection(A,B)`. It is called a symmetric difference because `ST_SymDifference(A,B) = ST_SymDifference`

If the optional `gridSize` argument is provided, the inputs are snapped to a grid of the given size, and the result vertices are computed on that same grid. (Requires GEOS-3.9.0 or higher)

Performed by the GEOS module

Enhanced: 3.1.0 accept a gridSize parameter.

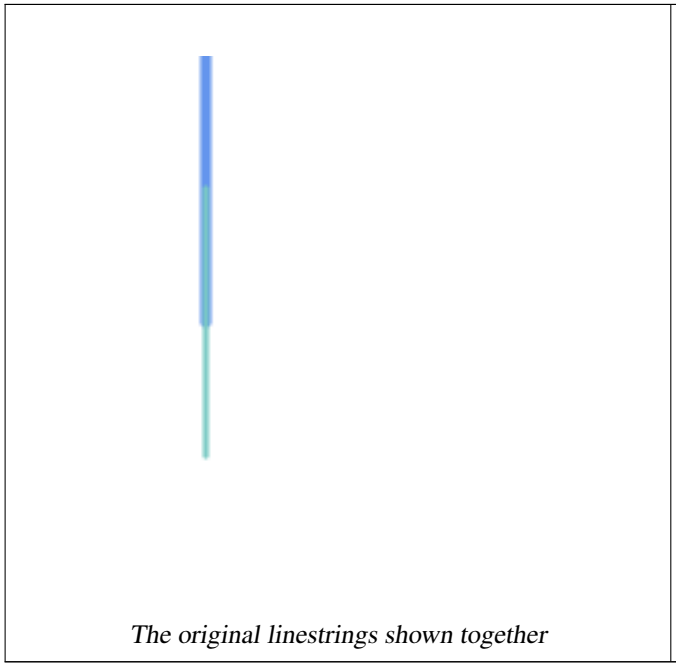
Requires GEOS >= 3.9.0 to use the gridSize parameter

This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.3

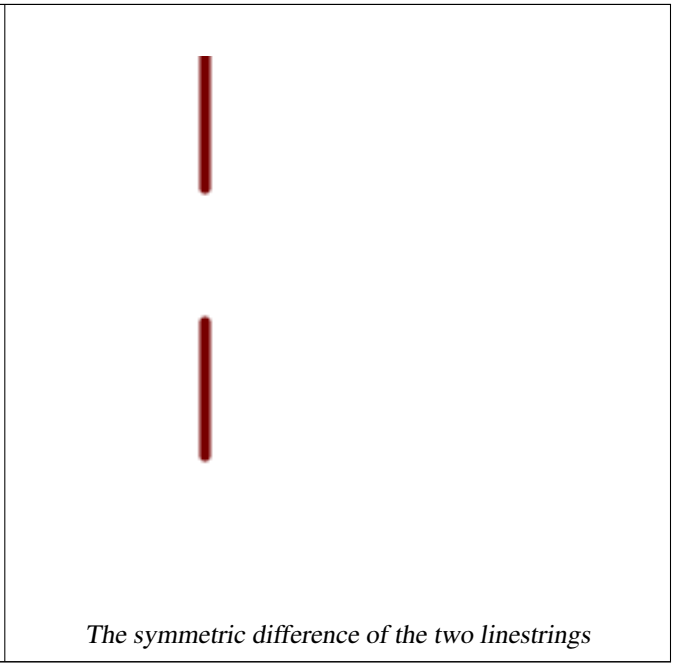
This method implements the SQL/MM specification. SQL-MM 3: 5.1.21

This function supports 3d and will not drop the z-index. However, the result is computed using XY only. The result Z values are copied, averaged or interpolated.

Examples



The original linestrings shown together



The symmetric difference of the two linestrings

```
--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)')
    )
);

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))
```

See Also

ST_Difference, ST_Intersection, ST_Union

7.13.9 ST_UnaryUnion

ST_UnaryUnion — Computes the union of the components of a single geometry.

Synopsis

geometry **ST_UnaryUnion**(geometry geom, float8 gridSize = -1);

Description

A single-input variant of ST_Union. The input may be a single geometry, a MultiGeometry, or a GeometryCollection. The union is applied to the individual elements of the input.

This function can be used to fix MultiPolygons which are invalid due to overlapping components. However, the input components must each be valid. An invalid input component such as a bow-tie polygon may cause an error. For this reason it may be better to use ST_MakeValid.

Another use of this function is to node and dissolve a collection of linestrings which cross or overlap to make them simple. (ST_Node also does this, but it does not provide the `gridSize` option.)

It is possible to combine ST_UnaryUnion with ST_GeomCollFromText to fine-tune how many geometries are be unioned at once. This allows trading off between memory usage and compute time, striking a balance between ST_Union and ST_MemUnion.

If the optional `gridSize` argument is provided, the inputs are snapped to a grid of the given size, and the result vertices are computed on that same grid. (Requires GEOS-3.9.0 or higher)

 This function supports 3d and will not drop the z-index. However, the result is computed using XY only. The result Z values are copied, averaged or interpolated.

Enhanced: 3.1.0 accept a gridSize parameter.

Requires GEOS >= 3.9.0 to use the gridSize parameter

Availability: 2.0.0

See Also

ST_Union, ST_MemUnion, ST_MakeValid, ST_GeomCollFromText, ST_Node

7.13.10 ST_Union

ST_Union — Computes a geometry representing the point-set union of the input geometries.

Synopsis

geometry **ST_Union**(geometry g1, geometry g2);
geometry **ST_Union**(geometry g1, geometry g2, float8 gridSize);
geometry **ST_Union**(geometry[] g1_array);
geometry **ST_Union**(geometry set g1field);
geometry **ST_Union**(geometry set g1field, float8 gridSize);

Description

Unions the input geometries, merging geometry to produce a result geometry with no overlaps. The output may be an atomic geometry, a MultiGeometry, or a Geometry Collection. Comes in several variants:

Two-input variant: returns a geometry that is the union of two input geometries. If either input is NULL, then NULL is returned.

Array variant: returns a geometry that is the union of an array of geometries.

Aggregate variant: returns a geometry that is the union of a rowset of geometries. The ST_Union() function is an "aggregate" function in the terminology of PostgreSQL. That means that it operates on rows of data, in the same way the SUM() and AVG() functions do and like most aggregates, it also ignores NULL geometries.

See ST_UnaryUnion for a non-aggregate, single-input variant.

The ST_Union array and set variants use the fast Cascaded Union algorithm described in http://blog.cleverelephant.ca/2009/01/-must-faster-unions-in-postgis-14.html

A `gridSize` can be specified to work in fixed-precision space. The inputs are snapped to a grid of the given size, and the result vertices are computed on that same grid. (Requires GEOS-3.9.0 or higher)

Note
ST_GeomCollFromText may sometimes be used in place of ST_Union, if the result is not required to be non-overlapping. ST_Collect is usually faster than ST_Union because it performs no processing on the collected geometries.

Performed by the GEOS module.

ST_Union creates MultiLineString and does not sew LineStrings into a single LineString. Use ST_LineMerge to sew LineStrings.

NOTE: this function was formerly called GeomUnion(), which was renamed from "Union" because UNION is an SQL reserved word.

Enhanced: 3.1.0 accept a gridSize parameter.

Requires GEOS >= 3.9.0 to use the gridSize parameter

Changed: 3.0.0 does not depend on SFCGAL.

Availability: 1.4.0 - ST_Union was enhanced. ST_Union(geomarray) was introduced and also faster aggregate collection in PostgreSQL.

 This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.3

Note
Aggregate version is not explicitly defined in OGC SPEC.

 This method implements the SQL/MM specification. SQL-MM 3: 5.1.19 the z-index (elevation) when polygons are involved.

 This function supports 3d and will not drop the z-index. However, the result is computed using XY only. The result Z values are copied, averaged or interpolated.

Examples

Aggregate example

```
SELECT id,
       ST_Union(geom) as singlegeom
FROM sometable f
GROUP BY id;
```

Non-Aggregate example

```
select ST_AsText(ST_Union('POINT(1 2)' :: geometry, 'POINT(-2 3)' :: geometry))

st_astext
----------
MULTIPOINT(-2 3,1 2)

select ST_AsText(ST_Union('POINT(1 2)' :: geometry, 'POINT(1 2)' :: geometry))

st_astext
----------
POINT(1 2)
```

3D example - sort of supports 3D (and with mixed dimensions!)

```
select ST_AsEWKT(ST_Union(geom))
from (
         select 'POLYGON((-7 4.2,-7.1 4.2,-7.1 4.3, -7 4.2))'::geometry geom
         union all
         select 'POINT(5 5 5)'::geometry geom
         union all
         select 'POINT(-2 3 1)'::geometry geom
         union all
         select 'LINESTRING(5 5 5, 10 10 10)'::geometry 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(geom))
from (
         select 'POLYGON((-7 4.2 2,-7.1 4.2 3,-7.1 4.3 2, -7 4.2 2))'::geometry geom
         union all
         select 'POINT(5 5 5)'::geometry geom
         union all
         select 'POINT(-2 3 1)'::geometry geom
         union all
         select 'LINESTRING(5 5 5, 10 10 10)'::geometry 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)))

--Examples using new Array construct
SELECT ST_Union(ARRAY(SELECT 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))
```

See Also

ST_GeomCollFromText, ST_UnaryUnion, ST_MemUnion, ST_Intersection, ST_Difference, ST_SymDifference

7.14 Processamento de Geometria

7.14.1 ST_Buffer

ST_Buffer — Computes a geometry covering all points within a given distance from a geometry.

Synopsis

geometry **ST_Buffer**(geometry g1, float radius_of_buffer, text buffer_style_parameters = '');
geometry **ST_Buffer**(geometry g1, float radius_of_buffer, integer num_seg_quarter_circle);
geography **ST_Buffer**(geography g1, float radius_of_buffer, text buffer_style_parameters);
geography **ST_Buffer**(geography g1, float radius_of_buffer, integer num_seg_quarter_circle);

Descrição

Computes a POLYGON or MULTIPOLYGON that represents all points whose distance from a geometry/geography is less than or equal to a given distance. A negative distance shrinks the geometry rather than expanding it. A negative distance may shrink a polygon completely, in which case POLYGON EMPTY is returned. For points and lines negative distances always return empty results.

For geometry, the distance is specified in the units of the Spatial Reference System of the geometry. For geography, the distance is specified in meters.

The optional third parameter controls the buffer accuracy and style. The accuracy of circular arcs in the buffer is specified as the number of line segments used to approximate a quarter circle (default is 8). The buffer style can be specifed by providing a list of blank-separated key=value pairs as follows:

- 'quad_segs=#' : number of line segments used to approximate a quarter circle (default is 8).
- 'endcap=round|flat|square' : endcap style (defaults to "round"). 'butt' is accepted as a synonym for 'flat'.
- 'join=round|mitre|bevel' : join style (defaults to "round"). 'miter' is accepted as a synonym for 'mitre'.
- 'mitre_limit=#.#' : mitre ratio limit (only affects mitered join style). 'miter_limit' is accepted as a synonym for 'mitre_limit'.
- 'side=both|left|right' : 'left' or 'right' performs a single-sided buffer on the geometry, with the buffered side relative to the direction of the line. This is only applicable to LINESTRING geometry and does not affect POINT or POLYGON geometries. By default end caps are square.

Note


For geography this is a thin wrapper around the geometry implementation. It determines a planar spatial reference system that best fits the bounding box of the geography object (trying UTM, Lambert Azimuthal Equal Area (LAEA) North/South pole, and finally Mercator). The buffer is computed in the planar space, and then transformed back to WGS84. This may not produce the desired behavior if the input object is much larger than a UTM zone or crosses the dateline

Note

Buffer output is always a valid polygonal geometry. Buffer can handle invalid inputs, so buffering by distance 0 is sometimes used as a way of repairing invalid polygons. ST_MakeValid can also be used for this purpose.

Note

Buffering is sometimes used to perform a within-distance search. For this use case it is more efficient to use ST_DWithin.

Note

This function ignores the Z dimension. It always gives a 2D result even when used on a 3D geometry.

Enhanced: 2.5.0 - ST_Buffer geometry support was enhanced to allow for side buffering specification `side=both|left|right`.

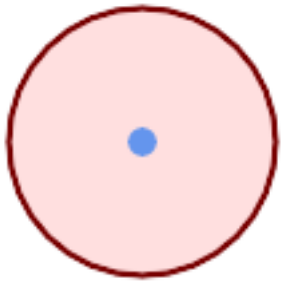
Availability: 1.5 - ST_Buffer was enhanced to support different endcaps and join types. These are useful for example to convert road linestrings into polygon roads with flat or square edges instead of rounded edges. Thin wrapper for geography was added.

Desempenhado pelo módulo GEOS.

This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.3

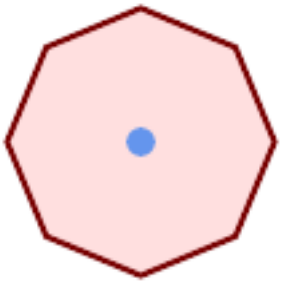
This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1.30

Exemplos



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');
```

endcap=round join=round (default)

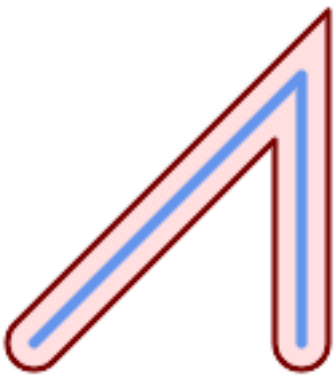
```
SELECT ST_Buffer(
 ST_GeomFromText(
  'LINESTRING(50 50,150 150,150 50)'
 ), 10, 'endcap=round join=round');
```

endcap=square

```
SELECT ST_Buffer(
 ST_GeomFromText(
  'LINESTRING(50 50,150 150,150 50)'
 ), 10, 'endcap=square join=round');
```

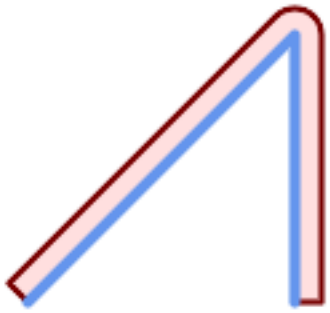
join=bevel

```
SELECT ST_Buffer(
 ST_GeomFromText(
  'LINESTRING(50 50,150 150,150 50)'
 ), 10, 'join=bevel');
```



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');
```

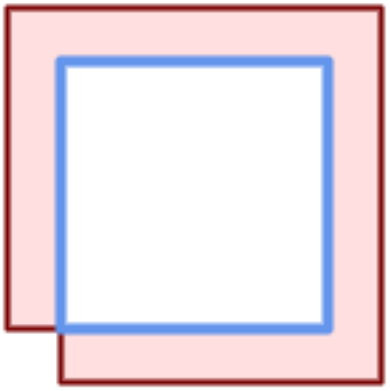


side=left

```
SELECT ST_Buffer(
 ST_GeomFromText(
  'LINESTRING(50 50,150 150,150 50)'
 ), 10, 'side=left');
```

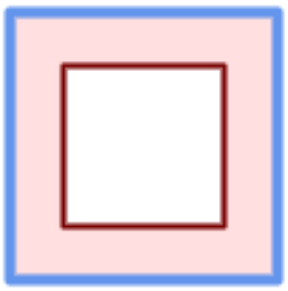
side=right

```
SELECT ST_Buffer(
 ST_GeomFromText(
  'LINESTRING(50 50,150 150,150 50)'
 ), 10, 'side=right');
```



right-hand-winding, polygon boundary side=left

```
SELECT ST_Buffer(
ST_ForceRHR(
ST_Boundary(
 ST_GeomFromText(
'POLYGON ((50 50, 50 150, 150 150, 150  ↩
    50, 50 50))'))),
 ), 20, 'side=left');
```



right-hand-winding, polygon boundary side=right

```
SELECT ST_Buffer(
ST_ForceRHR(
ST_Boundary(
 ST_GeomFromText(
'POLYGON ((50 50, 50 150, 150 150, 150  ↩
    50, 50 50))'))
), 20,'side=right')
```

```
--A buffered point approximates a circle
-- A buffered point forcing approximation of (see diagram)
```

```
-- 2 points per quarter 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
-- Note 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_Point(-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))
```

Veja também.

ST_GeomCollFromText, ST_DWithin, ST_SetSRID, ST_Transform, ST_Union, ST_MakeValid

7.14.2 ST_BuildArea

ST_BuildArea — Creates a polygonal geometry formed by the linework of a geometry.

Synopsis

geometry **ST_BuildArea**(geometry geom);

Descrição

Creates an areal geometry formed by the constituent linework of the input geometry. The input can be LINESTRINGS, MULTILINESTRINGS, POLYGONS, MULTIPOLYGONS, and GeometryCollections. The result is a Polygon or MultiPolygon, depending on input. If the input linework does not form polygons, NULL is returned.

This function assumes all inner geometries represent holes

Note
A linework de entrada deve ser nodificada corretamente para esta função trabalhar normalmente.

Disponibilidade: 1.1.0

Exemplos

These will create a donut

```
using polygons
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;

using linestrings
SELECT ST_BuildArea(ST_Collect(smallc,bigc))
FROM (SELECT
    ST_ExteriorRing(ST_Buffer(
      ST_GeomFromText('POINT(100 90)'), 25)) As smallc,
    ST_ExteriorRing(ST_Buffer(ST_GeomFromText('POINT(100 90)'), 50)) As bigc) As foo;
```

Veja também.

ST_Node, ST_MakePolygon, ST_MakeValid, ST_BdPolyFromText, ST_BdMPolyFromText (wrappers to this function with standard OGC interface)

7.14.3 ST_Centroid

ST_Centroid — Retorna o centro geométrico de uma geometria.

Synopsis

geometry **ST_Centroid**(geometry g1);
geography **ST_Centroid**(geography g1, boolean use_spheroid = true);

Descrição

Computes a point which is the geometric center of mass of a geometry. For [MULTI]POINTs, the centroid is the arithmetic mean of the input coordinates. For [MULTI]LINESTRINGs, the centroid is computed using the weighted length of each line segment. For [MULTI]POLYGONs, the centroid is computed in terms of area. If an empty geometry is supplied, an empty

GEOMETRYCOLLECTION is returned. If NULL is supplied, NULL is returned. If CIRCULARSTRING or COMPOUNDCURVE are supplied, they are converted to linestring with CurveToLine first, then same than for LINESTRING

For mixed-dimension input, the result is equal to the centroid of the component Geometries of highest dimension (since the lower-dimension geometries contribute zero "weight" to the centroid).

Note that for polygonal geometries the centroid does not necessarily lie in the interior of the polygon. For example, see the diagram below of the centroid of a C-shaped polygon. To construct a point guaranteed to lie in the interior of a polygon use ST_PointOnSurface.

New in 2.3.0 : supports CIRCULARSTRING and COMPOUNDCURVE (using CurveToLine)

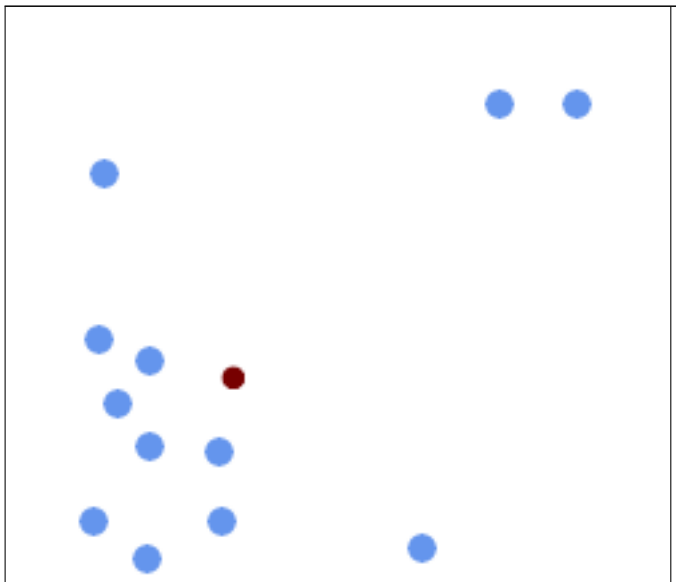
Availability: 2.4.0 support for geography was introduced.

This method implements the OGC Simple Features Implementation Specification for SQL 1.1.

This method implements the SQL/MM specification. SQL-MM 3: 8.1.4, 9.5.5

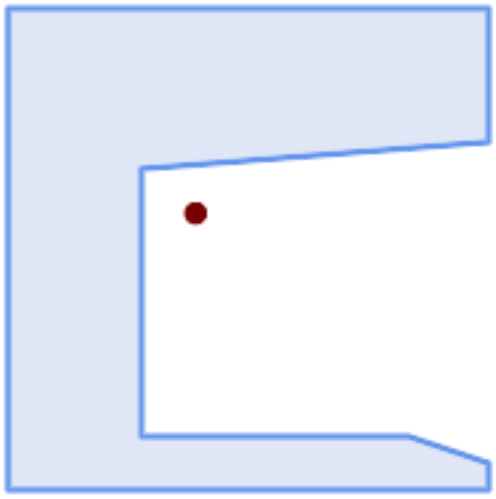
Exemplos

In the following illustrations the red dot is the centroid of the source geometry.

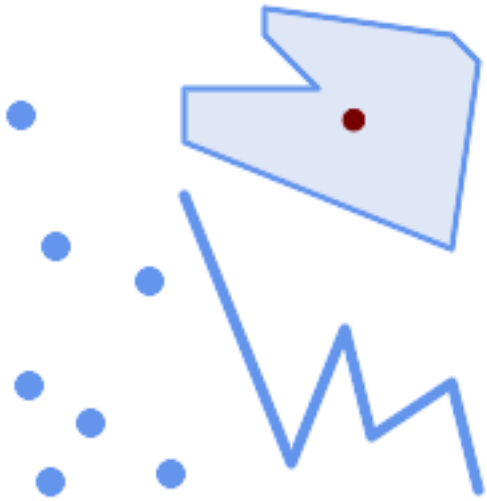


Centroide de um MULTIPOINT

Centroide de uma LINESTRING



Centroide de um `POLYGON`



Centroide de uma `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)

SELECT ST_AsText(ST_centroid(g))
FROM  ST_GeomFromText('CIRCULARSTRING(0 2, -1 1,0 0, 0.5 0, 1 0, 2 1, 1 2, 0.5 2, 0 2)')   ↩
    AS g ;
----------------------------------------
POINT(0.5 1)


SELECT ST_AsText(ST_centroid(g))
FROM  ST_GeomFromText('COMPOUNDCURVE(CIRCULARSTRING(0 2, -1 1,0 0),(0 0, 0.5 0, 1 0), ↩
    CIRCULARSTRING( 1 0, 2 1, 1 2),(1 2, 0.5 2, 0 2))' ) AS g;
----------------------------------------
POINT(0.5 1)
```

Veja também.

ST_PointOnSurface, ST_GeometricMedian

7.14.4 ST_ChaikinSmoothing

ST_ChaikinSmoothing — Returns a smoothed version of a geometry, using the Chaikin algorithm

Synopsis

geometry **ST_ChaikinSmoothing**(geometry geom, integer nIterations = 1, boolean preserveEndPoints = false);

Descrição

Smoothes a linear or polygonal geometry using Chaikin's algorithm. The degree of smoothing is controlled by the `nIterations` parameter. On each iteration, each interior vertex is replaced by two vertices located at 1/4 of the length of the line segments before and after the vertex. A reasonable degree of smoothing is provided by 3 iterations; the maximum is limited to 5.

If `preserveEndPoints` is true, the endpoints of Polygon rings are not smoothed. The endpoints of LineStrings are always preserved.

Note
The number of vertices doubles with each iteration, so the result geometry may have many more points than the input. To reduce the number of points use a simplification function on the result (see ST_Simplify, ST_SimplifyPreserveTopology and ST_SimplifyVW).

The result has interpolated values for the Z and M dimenions when present.

 This function supports 3d and will not drop the z-index.

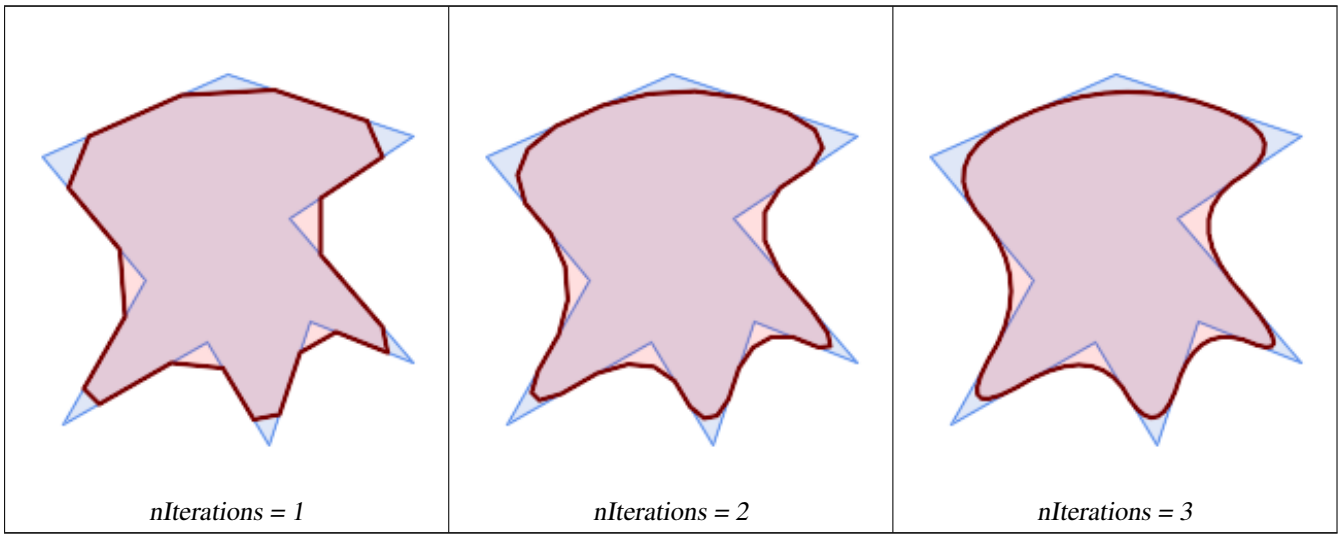
Availability: 2.5.0

Exemplos

Smoothing a triangle:

```
SELECT ST_AsText(ST_ChaikinSmoothing(geom)) smoothed
FROM (SELECT  'POLYGON((0 0, 8 8, 0 16, 0 0))'::geometry geom) AS foo;

                  smoothed
&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2500;&#x2
 POLYGON((2 2,6 6,6 10,2 14,0 12,0 4,2 2))
```

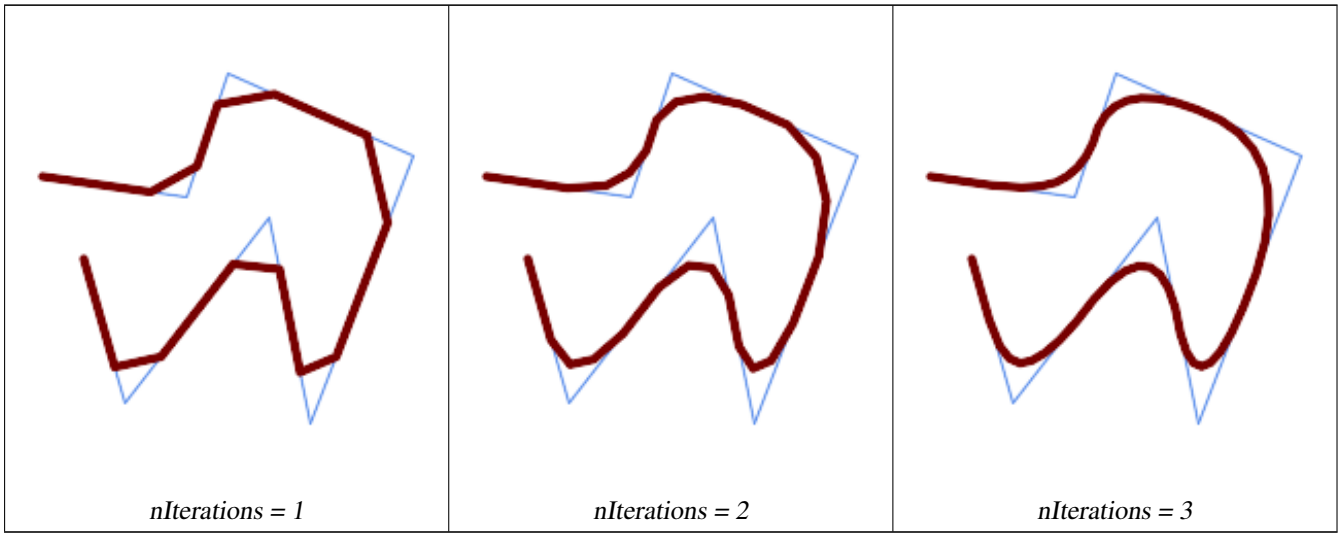
Smoothing a Polygon using 1, 2 and 3 iterations:



nIterations = 1 | *nIterations = 2* | *nIterations = 3*

```
SELECT ST_ChaikinSmoothing(
            'POLYGON ((20 20, 60 90, 10 150, 100 190, 190 160, 130 120, 190 50, 140 70, 120 ↩
                10, 90 60, 20 20))',
            generate_series(1, 3) );
```

Smoothing a LineString using 1, 2 and 3 iterations:



nIterations = 1 | *nIterations = 2* | *nIterations = 3*

```
SELECT ST_ChaikinSmoothing(
            'LINESTRING (10 140, 80 130, 100 190, 190 150, 140 20, 120 120, 50 30, 30 100) ↩
                ',
            generate_series(1, 3) );
```

Veja também.

ST_Simplify, ST_SimplifyPreserveTopology, ST_SimplifyVW

7.14.5 ST_ConcaveHull

ST_ConcaveHull — Computes a possibly concave geometry that contains all input geometry vertices

Synopsis

geometry **ST_ConcaveHull**(geometry param_geom, float param_pctconvex, boolean param_allow_holes = false);

Descrição

A concave hull is a (usually) concave geometry which contains the input, and whose vertices are a subset of the input vertices. In the general case the concave hull is a Polygon. The concave hull of two or more collinear points is a two-point LineString. The concave hull of one or more identical points is a Point. The polygon will not contain holes unless the optional `param_allow_holes` argument is specified as true.

One can think of a concave hull as "shrink-wrapping" a set of points. This is different to the convex hull, which is more like wrapping a rubber band around the points. A concave hull generally has a smaller area and represents a more natural boundary for the input points.

The `param_pctconvex` controls the concaveness of the computed hull. A value of 1 produces the convex hull. Values between 1 and 0 produce hulls of increasing concaveness. A value of 0 produces a hull with maximum concaveness (but still a single polygon). Choosing a suitable value depends on the nature of the input data, but often values between 0.3 and 0.1 produce reasonable results.

Note
Technically, the `param_pctconvex` determines a length as a fraction of the difference between the longest and shortest edges in the Delaunay Triangulation of the input points. Edges longer than this length are "eroded" from the triangulation. The triangles remaining form the concave hull.

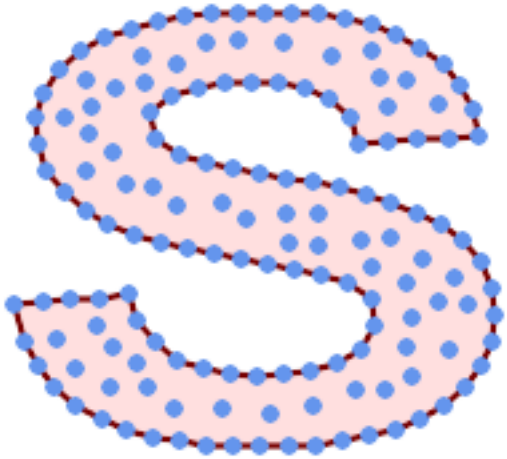
For point and linear inputs, the hull will enclose all the points of the inputs. For polygonal inputs, the hull will enclose all the points of the input *and also* all the areas covered by the input. If you want a point-wise hull of a polygonal input, convert it to points first using ST_Points.

This is not an aggregate function. To compute the concave hull of a set of geometries use ST_GeomCollFromText (e.g. `ST_ConcaveHull( ST_Collect( geom ), 0.80)`.

Disponibilidade: 2.0.0

Enhanced: 3.3.0, GEOS native implementation enabled for GEOS 3.11+

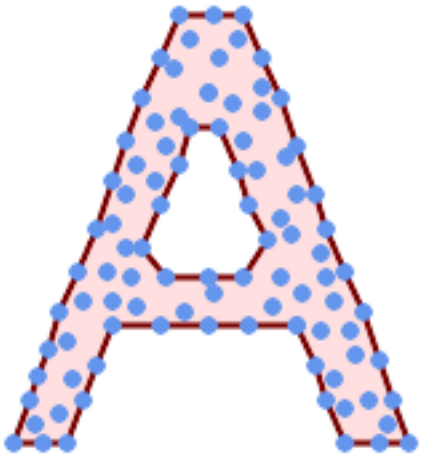
Exemplos



Concave Hull of a MultiPoint

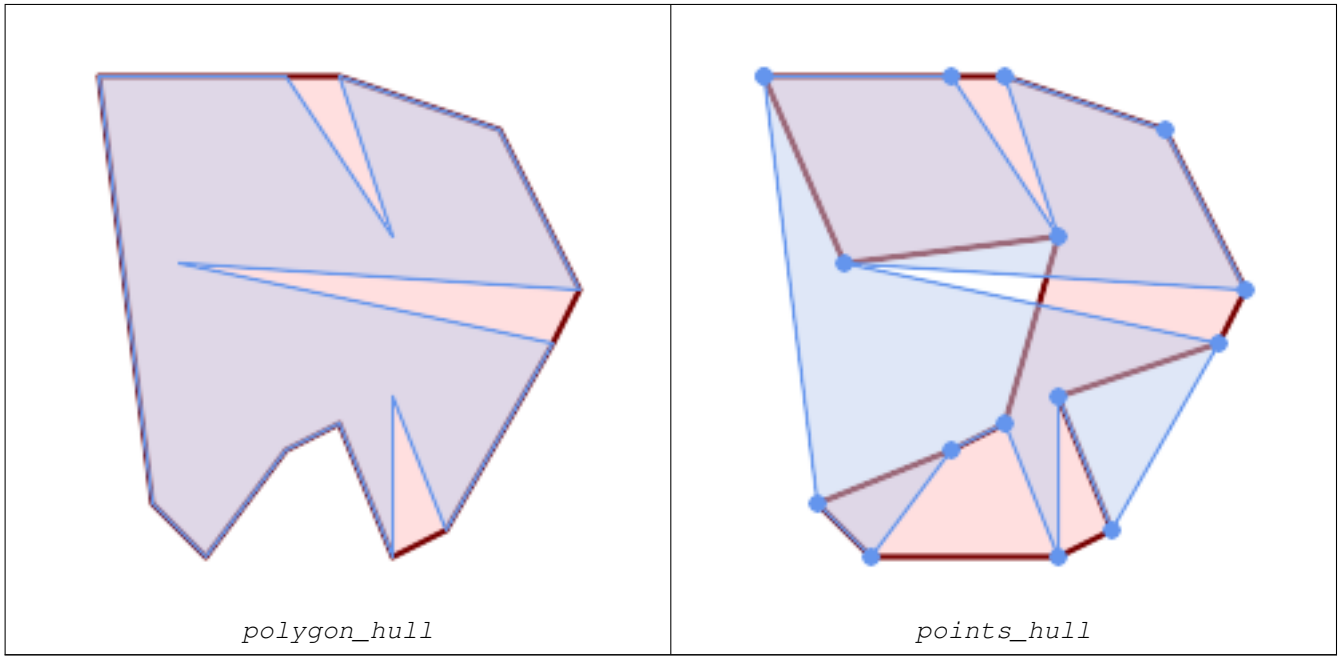
```
SELECT ST_AsText( ST_ConcaveHull(
        'MULTIPOINT ((10 72), (53 76), (56 66), (63 58), (71 51), (81 48), (91 46), (101  ↩
            45), (111 46), (121 47), (131 50), (140 55), (145 64), (144 74), (135 80), (125  ↩
            83), (115 85), (105 87), (95 89), (85 91), (75 93), (65 95), (55 98), (45 102),  ↩
            (37 107), (29 114), (22 122), (19 132), (18 142), (21 151), (27 160), (35 167),  ↩
            (44 172), (54 175), (64 178), (74 180), (84 181), (94 181), (104 181), (114 181) ↩
            , (124 181), (134 179), (144 177), (153 173), (162 168), (171 162), (177 154),  ↩
            (182 145), (184 135), (139 132), (136 142), (128 149), (119 153), (109 155), (99 ↩
             155), (89 155), (79 153), (69 150), (61 144), (63 134), (72 128), (82 125), (92 ↩
             123), (102 121), (112 119), (122 118), (132 116), (142 113), (151 110), (161  ↩
            106), (170 102), (178 96), (185 88), (189 78), (190 68), (189 58), (185 49),  ↩
            (179 41), (171 34), (162 29), (153 25), (143 23), (133 21), (123 19), (113 19),  ↩
            (102 19), (92 19), (82 19), (72 21), (62 22), (52 25), (43 29), (33 34), (25 41) ↩
            , (19 49), (14 58), (21 73), (31 74), (42 74), (173 134), (161 134), (150 133),  ↩
            (97 104), (52 117), (157 156), (94 171), (112 106), (169 73), (58 165), (149 40) ↩
            , (70 33), (147 157), (48 153), (140 96), (47 129), (173 55), (144 86), (159 67) ↩
            , (150 146), (38 136), (111 170), (124 94), (26 59), (60 41), (71 162), (41 64), ↩
             (88 110), (122 34), (151 97), (157 56), (39 146), (88 33), (159 45), (47 56),  ↩
            (138 40), (129 165), (33 48), (106 31), (169 147), (37 122), (71 109), (163 89), ↩
             (37 156), (82 170), (180 72), (29 142), (46 41), (59 155), (124 106), (157 80), ↩
             (175 82), (56 50), (62 116), (113 95), (144 167))',
```

```
        0.1 ) );
---st_astext--
POLYGON ((18 142, 21 151, 27 160, 35 167, 44 172, 54 175, 64 178, 74 180, 84 181, 94 181,  ↩
    104 181, 114 181, 124 181, 134 179, 144 177, 153 173, 162 168, 171 162, 177 154, 182  ↩
    145, 184 135, 173 134, 161 134, 150 133, 139 132, 136 142, 128 149, 119 153, 109 155, 99 ↩
     155, 89 155, 79 153, 69 150, 61 144, 63 134, 72 128, 82 125, 92 123, 102 121, 112 119,  ↩
    122 118, 132 116, 142 113, 151 110, 161 106, 170 102, 178 96, 185 88, 189 78, 190 68,  ↩
    189 58, 185 49, 179 41, 171 34, 162 29, 153 25, 143 23, 133 21, 123 19, 113 19, 102 19,  ↩
    92 19, 82 19, 72 21, 62 22, 52 25, 43 29, 33 34, 25 41, 19 49, 14 58, 10 72, 21 73, 31  ↩
    74, 42 74, 53 76, 56 66, 63 58, 71 51, 81 48, 91 46, 101 45, 111 46, 121 47, 131 50, 140 ↩
     55, 145 64, 144 74, 135 80, 125 83, 115 85, 105 87, 95 89, 85 91, 75 93, 65 95, 55 98,  ↩
    45 102, 37 107, 29 114, 22 122, 19 132, 18 142))
```



Concave Hull of a MultiPoint, allowing holes

```
SELECT ST_AsText( ST_ConcaveHull(
        'MULTIPOINT ((132 64), (114 64), (99 64), (81 64), (63 64), (57 49), (52 36), (46  ↩
            20), (37 20), (26 20), (32 36), (39 55), (43 69), (50 84), (57 100), (63 118),  ↩
            (68 133), (74 149), (81 164), (88 180), (101 180), (112 180), (119 164), (126  ↩
            149), (132 131), (139 113), (143 100), (150 84), (157 69), (163 51), (168 36),  ↩
            (174 20), (163 20), (150 20), (143 36), (139 49), (132 64), (99 151), (92 138),  ↩
            (88 124), (81 109), (74 93), (70 82), (83 82), (99 82), (112 82), (126 82), (121 ↩
             96), (114 109), (110 122), (103 138), (99 151), (34 27), (43 31), (48 44), (46  ↩
            58), (52 73), (63 73), (61 84), (72 71), (90 69), (101 76), (123 71), (141 62),  ↩
            (166 27), (150 33), (159 36), (146 44), (154 53), (152 62), (146 73), (134 76),  ↩
            (143 82), (141 91), (130 98), (126 104), (132 113), (128 127), (117 122), (112  ↩
            133), (119 144), (108 147), (119 153), (110 171), (103 164), (92 171), (86 160), ↩
             (88 142), (79 140), (72 124), (83 131), (79 118), (68 113), (63 102), (68 93),  ↩
            (35 45))',
         0.15, true ) );
---st_astext--
POLYGON ((43 69, 50 84, 57 100, 63 118, 68 133, 74 149, 81 164, 88 180, 101 180, 112 180,  ↩
    119 164, 126 149, 132 131, 139 113, 143 100, 150 84, 157 69, 163 51, 168 36, 174 20, 163 ↩
     20, 150 20, 143 36, 139 49, 132 64, 114 64, 99 64, 81 64, 63 64, 57 49, 52 36, 46 20,  ↩
    37 20, 26 20, 32 36, 35 45, 39 55, 43 69), (88 124, 81 109, 74 93, 83 82, 99 82, 112 82, ↩
     121 96, 114 109, 110 122, 103 138, 92 138, 88 124))
```



Comparing a concave hull of a Polygon to the concave hull of the constituent points. The hull respects the boundary of the polygon, whereas the points-based hull does not.

```
WITH data(geom) AS (VALUES
   ('POLYGON ((10 90, 39 85, 61 79, 50 90, 80 80, 95 55, 25 60, 90 45, 70 16, 63 38, 60 10, ↩
       50 30, 43 27, 30 10, 20 20, 10 90))'::geometry)
)
SELECT  ST_ConcaveHull( geom,            0.1) AS polygon_hull,
        ST_ConcaveHull( ST_Points(geom), 0.1) AS points_hull
    FROM data;
```

Using with ST_Collect to compute the concave hull of a geometry set.

```
-- Compute estimate of infected area based on point observations
SELECT disease_type,
    ST_ConcaveHull( ST_Collect(obs_pnt), 0.3 ) AS geom
  FROM disease_obs
  GROUP BY disease_type;
```

Veja também.

ST_ConvexHull, ST_GeomCollFromText, ST_AlphaShape, ST_OptimalAlphaShape

7.14.6 ST_ConvexHull

ST_ConvexHull — Computes the convex hull of a geometry.

Synopsis

geometry **ST_ConvexHull**(geometry geomA);

Descrição

Computes the convex hull of a geometry. The convex hull is the smallest convex geometry that encloses all geometries in the input.

One can think of the convex hull as the geometry obtained by wrapping an rubber band around a set of geometries. This is different from a concave hull which is analogous to "shrink-wrapping" the geometries. A convex hull is often used to determine an affected area based on a set of point observations.

In the general case the convex hull is a Polygon. The convex hull of two or more collinear points is a two-point LineString. The convex hull of one or more identical points is a Point.

This is not an aggregate function. To compute the convex hull of a set of geometries, use ST_GeomCollFromText to aggregate them into a geometry collection (e.g. `ST_ConvexHull(ST_Collect(geom))`.

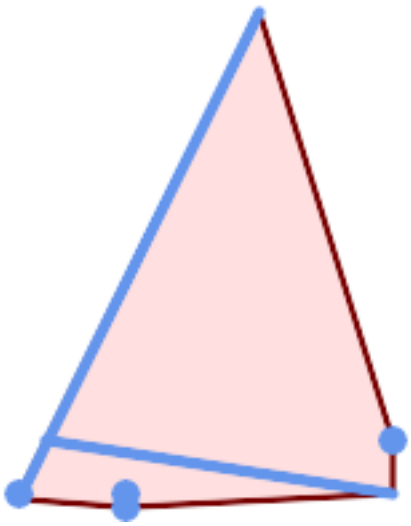
Desempenhado pelo módulo GEOS

This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.3

This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1.16

This function supports 3d and will not drop the z-index.

Exemplos



Convex Hull of a MultiLinestring and a 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))
```

Using with ST_Collect to compute the convex hulls of geometry sets.

```
--Get estimate of infected area based on point observations
SELECT d.disease_type,
    ST_ConvexHull(ST_Collect(d.geom)) As geom
    FROM disease_obs As d
    GROUP BY d.disease_type;
```

Veja também.

ST_GeomCollFromText, ST_ConcaveHull, ST_MinimumBoundingCircle

7.14.7 ST_DelaunayTriangles

ST_DelaunayTriangles — Returns the Delaunay triangulation of the vertices of a geometry.

Synopsis

geometry **ST_DelaunayTriangles**(geometry g1, float tolerance = 0.0, int4 flags = 0);

Descrição

Computes the Delaunay triangulation of the vertices of the input geometry. The optional `tolerance` can be used to snap nearby input vertices together, which improves robustness in some situations. The result geometry is bounded by the convex hull of the input vertices. The result geometry representation is determined by the `flags` code:

- `0` - a GEOMETRYCOLLECTION of triangular POLYGONs (default)
- `1` - a MULTILINESTRING of the edges of the triangulation
- `2` - A TIN of the triangulation

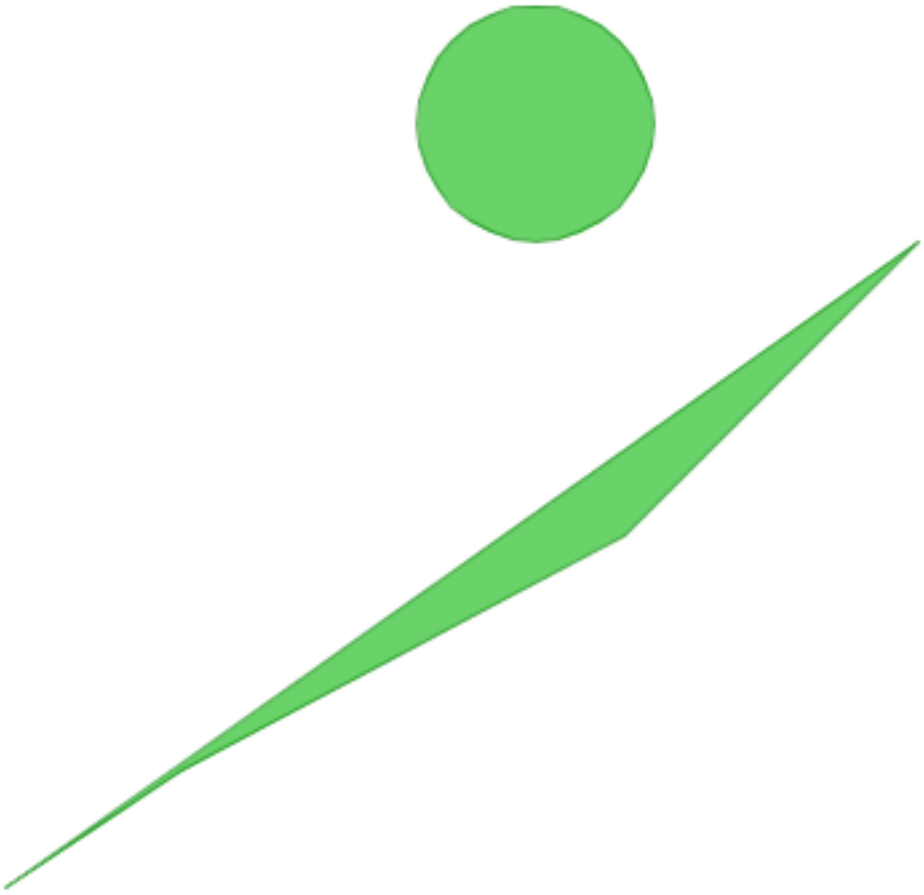
Desempenhado pelo módulo GEOS.

Disponibilidade: 2.1.0

 This function supports 3d and will not drop the z-index.

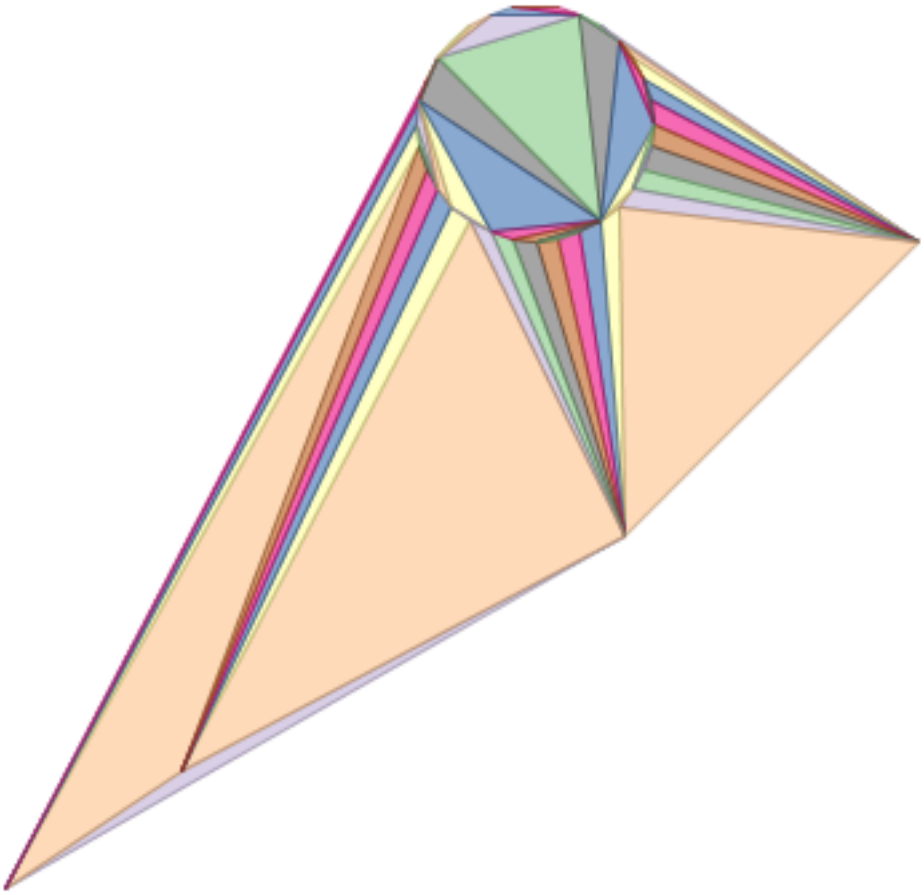
 This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

Exemplos



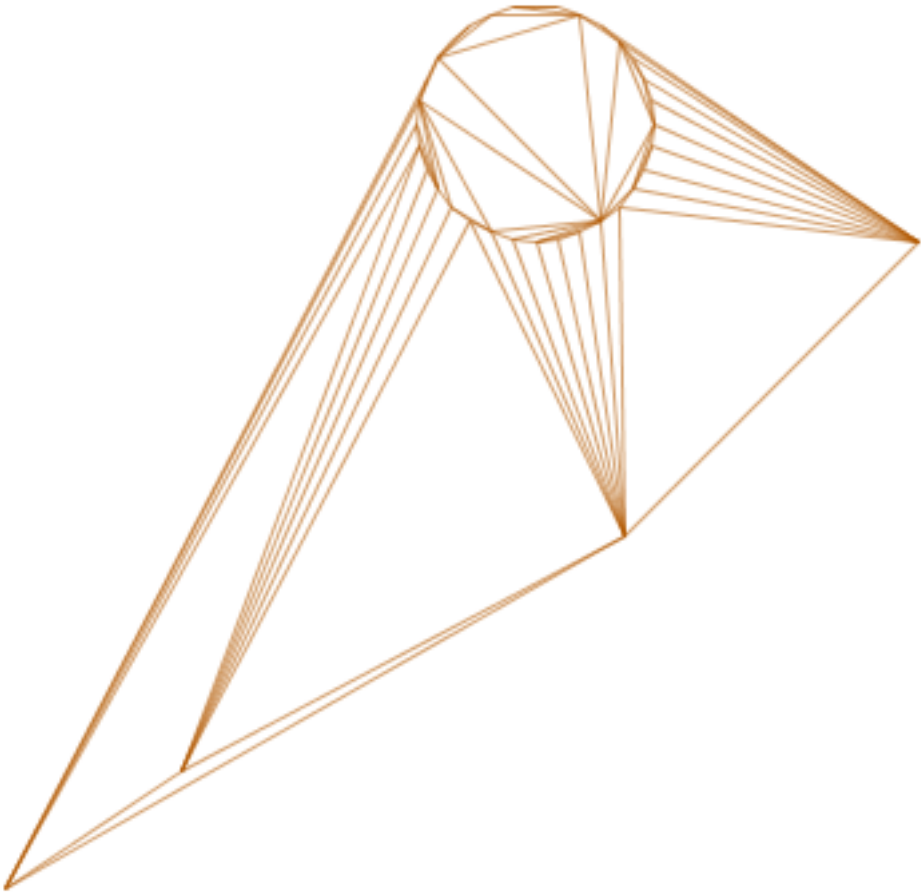
Polígonos originais

```
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)
        )
```



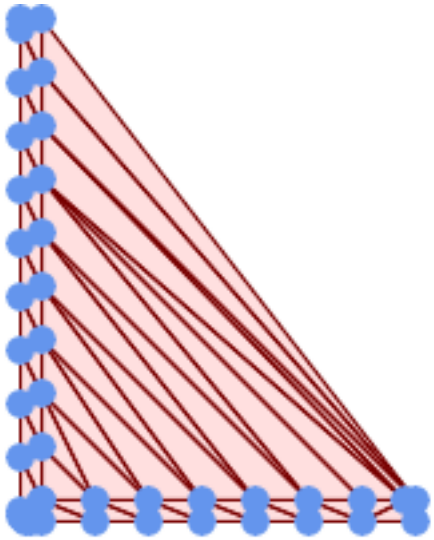
ST_DelaunayTriangles de 2 polígonos: polígonos da triangulação de Delaunay, cada triângulo de uma cor diferente

```
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;
```



-- triângulos de Delaunay como 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;
```



-- triângulos delaunay de 45 pontos como 55 polígonos triangulares

```
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))
```

Example using vertices with Z values.

```
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)))
```

Veja também.

ST_VoronoiPolygons, ST_TriangulatePolygon, ST_ConstrainedDelaunayTriangles, ST_VoronoiLines, ST_ConvexHull

7.14.8 ST_FilterByM

ST_FilterByM — Removes vertices based on their M value

Synopsis

geometry **ST_FilterByM**(geometry geom, double precision min, double precision max = null, boolean returnM = false);

Descrição

Filters out vertex points based on their M-value. Returns a geometry with only vertex points that have a M-value larger or equal to the min value and smaller or equal to the max value. If max-value argument is left out only min value is considered. If fourth argument is left out the m-value will not be in the resulting geometry. If resulting geometry have too few vertex points left for its geometry type an empty geometry will be returned. In a geometry collection geometries without enough points will just be left out silently.

This function is mainly intended to be used in conjunction with ST_SetEffectiveArea. ST_EffectiveArea sets the effective area of a vertex in its m-value. With ST_FilterByM it then is possible to get a simplified version of the geometry without any calculations, just by filtering

Note
There is a difference in what ST_SimplifyVW returns when not enough points meet the criteria compared to ST_FilterByM. ST_SimplifyVW returns the geometry with enough points while ST_FilterByM returns an empty geometry

Note
Note that the returned geometry might be invalid

Note
This function returns all dimensions, including the Z and M values

Availability: 2.5.0

Exemplos

A linestring is filtered

```
SELECT ST_AsText(ST_FilterByM(geom,30)) simplified
FROM (SELECT  ST_SetEffectiveArea('LINESTRING(5 2, 3 8, 6 20, 7 25, 10 10)'::geometry) geom ↩
    ) As foo;

result

         simplified
----------------------------
 LINESTRING(5 2,7 25,10 10)
```

Veja também.

ST_SetEffectiveArea, ST_SimplifyVW

7.14.9 ST_GeneratePoints

ST_GeneratePoints — Generates random points contained in a Polygon or MultiPolygon.

Synopsis

geometry **ST_GeneratePoints**(g geometry , npoints integer);
geometry **ST_GeneratePoints**(geometry g , integer npoints , integer seed = 0);

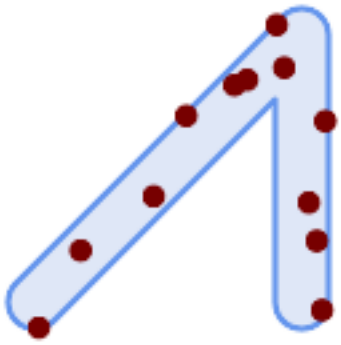
Descrição

ST_GeneratePoints generates a given number of pseudo-random points which lie within the input area. The optional `seed` is used to regenerate a deterministic sequence of points, and must be greater than zero.

Disponibilidade: 2.3.0

Enhanced: 3.0.0, added seed parameter

Exemplos



Generated 12 Points overlaid on top of original polygon using a random seed value 1996

```
SELECT ST_GeneratePoints(geom, 12, 1996)
FROM (
    SELECT ST_Buffer(
        ST_GeomFromText(
        'LINESTRING(50 50,150 150,150 50)'),
        10, 'endcap=round join=round') AS geom
) AS s;
```

7.14.10 ST_GeometricMedian

ST_GeometricMedian — Retorna a mediana de um MultiPonto.

Synopsis

geometry **ST_GeometricMedian** (geometry geom, float8 tolerance = NULL, int max_iter = 10000, boolean fail_if_not_converged = false);

Descrição

Computes the approximate geometric median of a MultiPoint geometry using the Weiszfeld algorithm. The geometric median is the point minimizing the sum of distances to the input points. It provides a centrality measure that is less sensitive to outlier points than the centroid (center of mass).

The algorithm iterates until the distance change between successive iterations is less than the supplied `tolerance` parameter. If this condition has not been met after `max_iterations` iterations, the function produces an error and exits, unless `fail_if_not_converged` is set to `false` (the default).

If a `tolerance` argument is not provided, the tolerance value is calculated based on the extent of the input geometry.

If present, the input point M values are interpreted as their relative weights.

Disponibilidade: 2.3.0

Enhanced: 2.5.0 Added support for M as weight of points.

 This function supports 3d and will not drop the z-index.

 This function supports M coordinates.

Exemplos

Comparison of the geometric median (red) and centroid (turquoise) of a MultiPoint.

```
WITH test AS (
SELECT 'MULTIPOINT((10 10), (10 40), (40 10), (190 190))'::geometry geom)
SELECT
  ST_AsText(ST_Centroid(geom)) centroid,
  ST_AsText(ST_GeometricMedian(geom)) median
FROM test;

      centroid      |                 median
--------------------+----------------------------------------
   POINT(62.5 62.5) | POINT(25.01778421249728 25.01778421249728)
(1 row)
```

Veja também.

ST_Centroid

7.14.11 ST_LineMerge

ST_LineMerge — Return the lines formed by sewing together a MultiLineString.

Synopsis

geometry **ST_LineMerge**(geometry amultilinestring);
geometry **ST_LineMerge**(geometry amultilinestring, boolean directed);

Descrição

Returns a LineString or MultiLineString formed by joining together the line elements of a MultiLineString. Lines are joined at their endpoints at 2-way intersections. Lines are not joined across intersections of 3-way or greater degree.

If **directed** is TRUE, then ST_LineMerge will not change point order within LineStrings, so lines with opposite directions will not be merged

Note
Only use with MultiLineString/LineStrings. Other geometry types return an empty GeometryCollection

Desempenhado pelo módulo GEOS.

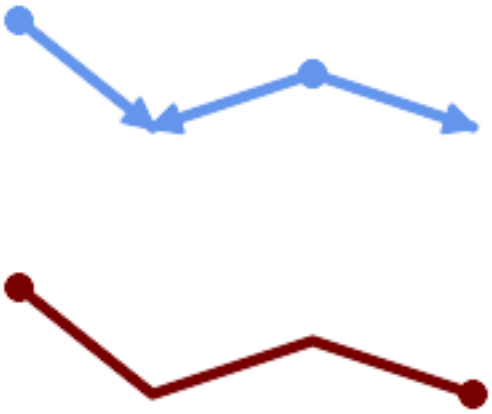
Enhanced: 3.3.0 accept a directed parameter.

Requires GEOS >= 3.11.0 to use the directed parameter.

Disponibilidade: 1.1.0

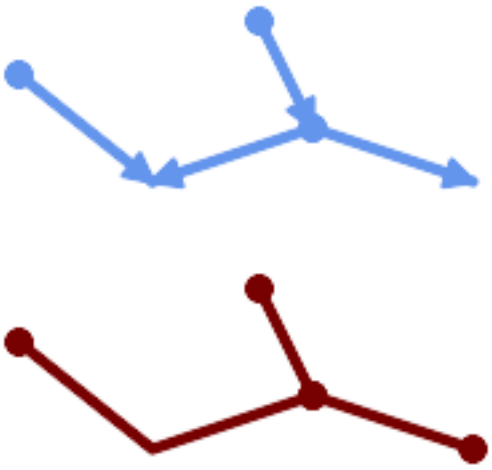
Warning
This function strips the M dimension.

Exemplos



Merging lines with different orientation.

```
SELECT ST_AsText(ST_LineMerge(
'MULTILINESTRING((10 160, 60 120), (120 140, 60 120), (120 140, 180 120))'
                ));
-----------------------------------------
 LINESTRING(10 160,60 120,120 140,180 120)
```

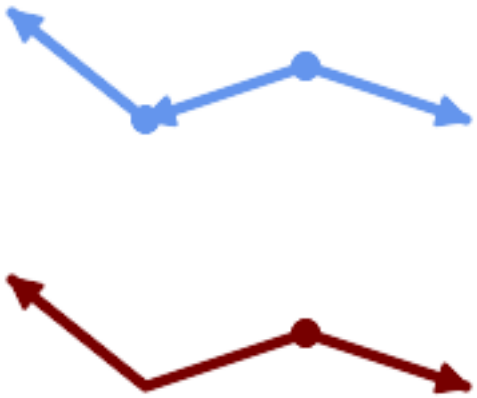


Lines are not merged across intersections with degree > 2.

```
SELECT ST_AsText(ST_LineMerge(
'MULTILINESTRING((10 160, 60 120), (120 140, 60 120), (120 140, 180 120), (100 180, 120  ↩
    140))'
                ));
-----------------------------------------
 MULTILINESTRING((10 160,60 120,120 140),(100 180,120 140),(120 140,180 120))
```

If merging is not possible due to non-touching lines, the original MultiLineString is returned.

```
SELECT ST_AsText(ST_LineMerge(
'MULTILINESTRING((-29 -27,-30 -29.7,-36 -31,-45 -33),(-45.2 -33.2,-46 -32))'
));
----------------
MULTILINESTRING((-45.2 -33.2,-46 -32),(-29 -27,-30 -29.7,-36 -31,-45 -33))
```



Lines with opposite directions are not merged if directed = TRUE.

```
SELECT ST_AsText(ST_LineMerge(
'MULTILINESTRING((60 30, 10 70), (120 50, 60 30), (120 50, 180 30))',
TRUE));
----------------------------------------------------
 MULTILINESTRING((120 50,60 30,10 70),(120 50,180 30))
```

Example showing Z-dimension handling.

```
SELECT ST_AsText(ST_LineMerge(
      'MULTILINESTRING((-29 -27 11,-30 -29.7 10,-36 -31 5,-45 -33 6), (-29 -27 12,-30 -29.7 ↩
          5), (-45 -33 1,-46 -32 11))'
        ));
------------------------------------------------------------------------------------------------ ←

LINESTRING Z (-30 -29.7 5,-29 -27 11,-30 -29.7 10,-36 -31 5,-45 -33 1,-46 -32 11)
```

Veja também.

ST_Segmentize, ST_LineSubstring

7.14.12 ST_MaximumInscribedCircle

ST_MaximumInscribedCircle — Retorna o centro geométrico de uma geometria.

Synopsis

(geometry, geometry, double precision) **ST_MaximumInscribedCircle**(geometry geom);

Descrição

Finds the largest circle that is contained within a (multi)polygon, or which does not overlap any lines and points. Returns a record with fields:

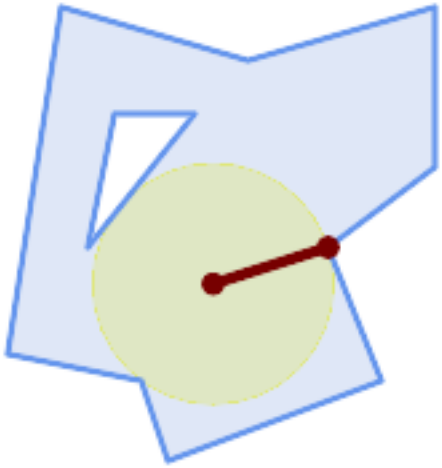
- `center` - center point of the circle
- `nearest` - a point on the geometry nearest to the center
- `radius` - radius of the circle

For polygonal inputs, the circle is inscribed within the boundary rings, using the internal rings as boundaries. For linear and point inputs, the circle is inscribed within the convex hull of the input, using the input lines and points as further boundaries.

Availability: 3.1.0.

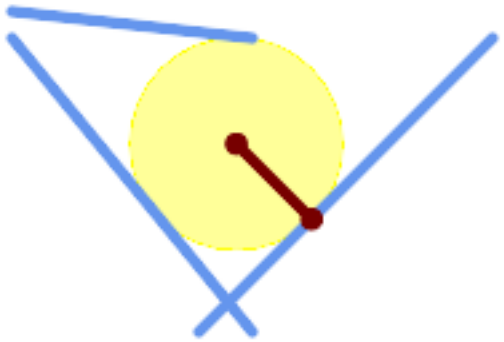
Requires GEOS >= 3.9.0.

Exemplos



Maximum inscribed circle of a polygon. Center, nearest point, and radius are returned.

```
SELECT radius, ST_AsText(center) AS center, ST_AsText(nearest) AS nearest
    FROM ST_MaximumInscribedCircle(
        'POLYGON ((40 180, 110 160, 180 180, 180 120, 140 90, 160 40, 80 10, 70 40, 20 50,  ↩
            40 180),
        (60 140, 50 90, 90 140, 60 140))');

     radius      |           center           |    nearest
-----------------+----------------------------+---------------
 45.165845650018 | POINT(96.953125 76.328125) | POINT(140 90)
```



Maximum inscribed circle of a multi-linestring. Center, nearest point, and radius are returned.

Veja também.

ST_MinimumBoundingRadius, ST_LargestEmptyCircle

7.14.13 ST_LargestEmptyCircle

ST_LargestEmptyCircle — Computes the largest circle not overlapping a geometry.

Synopsis

(geometry, geometry, double precision) **ST_LargestEmptyCircle**(geometry geom, double precision tolerance=0.0, geometry boundary=POINT EMPTY);

Descrição

Finds the largest circle which does not overlap a set of point and line obstacles. (Polygonal geometries may be included as obstacles, but only their boundary lines are used.) The center of the circle is constrained to lie inside a polygonal boundary, which by default is the convex hull of the input geometry. The circle center is the point in the interior of the boundary which has the farthest distance from the obstacles. The circle itself is provided by the center point and a nearest point lying on an obstacle detemining the circle radius.

The circle center is determined to a given accuracy specified by a distance tolerance, using an iterative algorithm. If the accuracy distance is not specified a reasonable default is used.

Returns a record with fields:

- `center` - center point of the circle
- `nearest` - a point on the geometry nearest to the center
- `radius` - radius of the circle

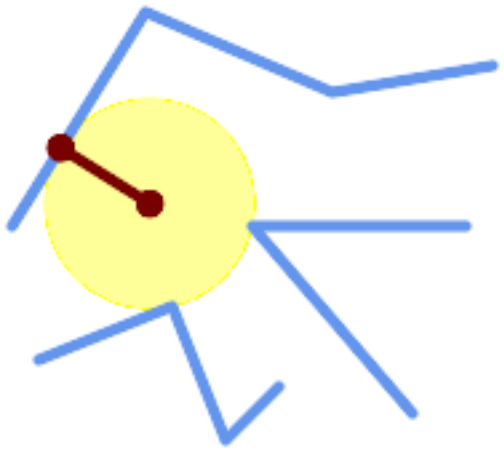
To find the largest empty circle in the interior of a polygon, see ST_MaximumInscribedCircle.

Availability: 3.4.0.

Requires GEOS >= 3.9.0.

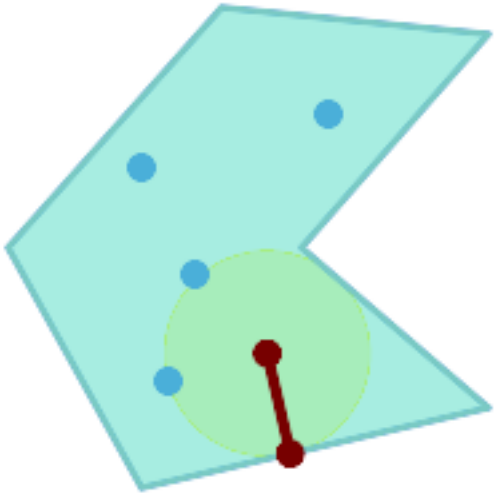
Exemplos

```
SELECT radius,
       ST_AsText(center) AS center,
       ST_AsText(nearest) AS nearest
  FROM ST_LargestEmptyCircle(
        'MULTILINESTRING (
          (10 100, 60 180, 130 150, 190 160),
          (20 50, 70 70, 90 20, 110 40),
          (160 30, 100 100, 180 100))');
```



Largest Empty Circle within a set of lines.

```
SELECT radius,
       ST_AsText(center) AS center,
       ST_AsText(nearest) AS nearest
  FROM ST_LargestEmptyCircle(
         St_Collect(
           'MULTIPOINT ((70 50), (60 130), (130 150), (80 90))',
           'POLYGON ((90 190, 10 100, 60 10, 190 40, 120 100, 190 180, 90 190))'),
         'POLYGON ((90 190, 10 100, 60 10, 190 40, 120 100, 190 180, 90 190))'
       );
```



Largest Empty Circle within a set of points, constrained to lie in a polygon. The constraint polygon boundary must be included as an obstacle, as well as specified as the constraint for the circle center.

Veja também.

ST_MinimumBoundingRadius

7.14.14 ST_MinimumBoundingCircle

ST_MinimumBoundingCircle — Returns the smallest circle polygon that contains a geometry.

Synopsis

geometry **ST_MinimumBoundingCircle**(geometry geomA, integer num_segs_per_qt_circ=48);

Descrição

Returns the smallest circle polygon that contains a geometry.

Note
O círculo é aproximado por um polígono com um padrão de 48 segmentos por quarto de círculo. Devido ao polígono ser uma aproximação do círculo delimitador, alguns pontos na geometria de entrada podem não estar contidos dentro do polígono. A aproximação pode ser melhorada pelo aumento do número de segmentos, com uma pequena penalidade de desempenho. Para aplicações nas quais uma aproximação poligonal não se encaixa, a ST_MinimumBoundingRadius pode ser usada.

Use with ST_GeomCollFromText to get the minimum bounding circle of a set of geometries.

To compute two points lying on the mimimum circle (the "maximum diameter") use ST_LongestLine.

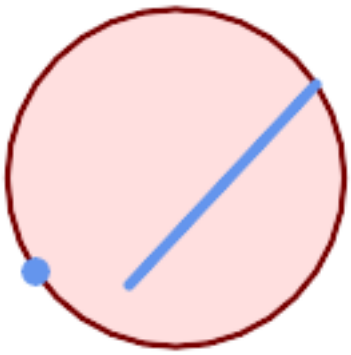
A razão da área de um polígono dividido pela área do seu menor círculo delimitador é referenciada com o teste Roeck.

Desempenhado pelo módulo GEOS.

Disponibilidade: 1.4.0

Exemplos

```
SELECT d.disease_type,
    ST_MinimumBoundingCircle(ST_Collect(d.geom)) As geom
    FROM disease_obs As d
    GROUP BY d.disease_type;
```



Menor círculo delimitador de um ponto e linestring. Utilizando 8 segmentos para aproximar um quarto de círculo

```
SELECT ST_AsText(ST_MinimumBoundingCircle(
        ST_Collect(
            ST_GeomFromText('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))
```

Veja também.

ST_GeomCollFromText, ST_MinimumBoundingRadius, ST_LargestEmptyCircle, ST_LongestLine

7.14.15 ST_MinimumBoundingRadius

ST_MinimumBoundingRadius — Returns the center point and radius of the smallest circle that contains a geometry.

Synopsis

(geometry, double precision) **ST_MinimumBoundingRadius**(geometry geom);

Descrição

Computes the center point and radius of the smallest circle that contains a geometry. Returns a record with fields:

- `center` - center point of the circle
- `radius` - radius of the circle

Use with ST_GeomCollFromText to get the minimum bounding circle of a set of geometries.

To compute two points lying on the mimimum circle (the "maximum diameter") use ST_LongestLine.

Disponibilidade - 2.3.0

Exemplos

```
SELECT ST_AsText(center), radius FROM ST_MinimumBoundingRadius('POLYGON((26426 65078,26531 ↩
    65242,26075 65136,26096 65427,26426 65078))');

                st_astext                 |      radius
------------------------------------------+------------------
 POINT(26284.8418027133 65267.1145090825) | 247.436045591407
```

Veja também.

ST_GeomCollFromText, ST_MinimumBoundingCircle, ST_LongestLine

7.14.16 ST_OrientedEnvelope

ST_OrientedEnvelope — Returns a minimum-area rectangle containing a geometry.

Synopsis

geometry **ST_OrientedEnvelope**(geometry geom);

Descrição

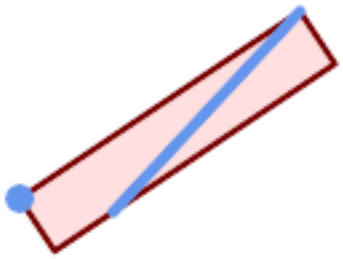
Returns the minimum-area rotated rectangle enclosing a geometry. Note that more than one such rectangle may exist. May return a Point or LineString in the case of degenerate inputs.

Availability: 2.5.0.

Requires GEOS >= 3.6.0.

Exemplos

```
SELECT ST_AsText(ST_OrientedEnvelope('MULTIPOINT ((0 0), (-1 -1), (3 2))'));

                st_astext
                -----------------------------------------------
                POLYGON((3 2,2.88 2.16,-1.12 -0.84,-1 -1,3 2))
```



Oriented envelope of a point and linestring.

```
SELECT ST_AsText(ST_OrientedEnvelope(
        ST_Collect(
            ST_GeomFromText('LINESTRING(55 75,125 150)'),
                ST_Point(20, 80))
                )) As wktenv;
wktenv
-----------
POLYGON((19.9999999999997 79.9999999999999,33.0769230769229  ↩
    60.3846153846152,138.076923076924 130.384615384616,125.000000000001  ↩
    150.000000000001,19.9999999999997 79.9999999999999))
```

Veja também.

ST_Envelope ST_MinimumBoundingCircle

7.14.17 ST_OffsetCurve

ST_OffsetCurve — Returns an offset line at a given distance and side from an input line.

Synopsis

geometry **ST_OffsetCurve**(geometry line, float signed_distance, text style_parameters='');

Descrição

Return an offset line at a given distance and side from an input line. All points of the returned geometries are not further than the given distance from the input geometry. Useful for computing parallel lines about a center line.

For positive distance the offset is on the left side of the input line and retains the same direction. For a negative distance it is on the right side and in the opposite direction.

Unidades de distância são medidas em unidades do sistema de referência espacial.

Note that output may be a MULTILINESTRING or EMPTY for some jigsaw-shaped input geometries.

O terceiro parâmetro opcional permite especificar uma lista de chave=valor em branco separados em pares para ajustar operações como segue:

- 'quad_segs=#' : número de segmentos usado para aproximar um quarto de círculo (leva a 8).
- 'join=round|mitre|bevel' : join style (padrão para "round"). 'miter' também é aceitado como sinônimo de 'mitre'.
- 'mitre_limit=#.#' : mitre ratio limit (só afeta o estilo mitred join). 'miter_limit' também é aceito como sinônimo para 'mitre_limit'.

Desempenhado pelo módulo GEOS.

Disponibilidade: 2.0

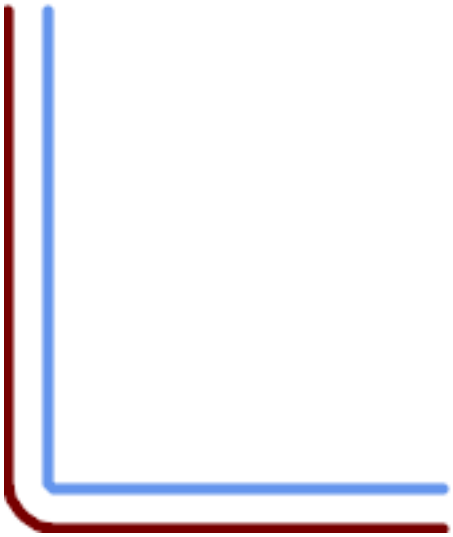
Enhanced: 2.5 - added support for GEOMETRYCOLLECTION and MULTILINESTRING

Note
This function ignores the Z dimension. It always gives a 2D result even when used on a 3D geometry.

Exemplos

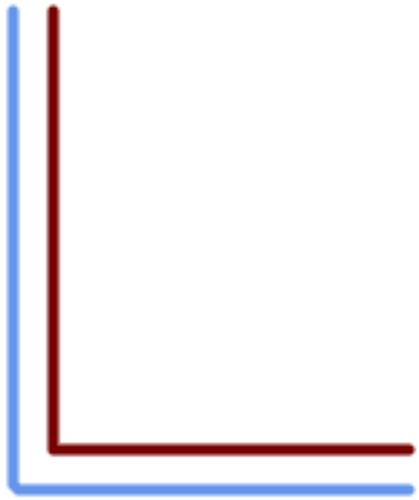
Calcula um buffer aberto em volta das ruas

```
SELECT ST_Union(
 ST_OffsetCurve(f.geom,  f.width/2, 'quad_segs=4 join=round'),
 ST_OffsetCurve(f.geom, -f.width/2, 'quad_segs=4 join=round')
) as track
FROM someroadstable;
```



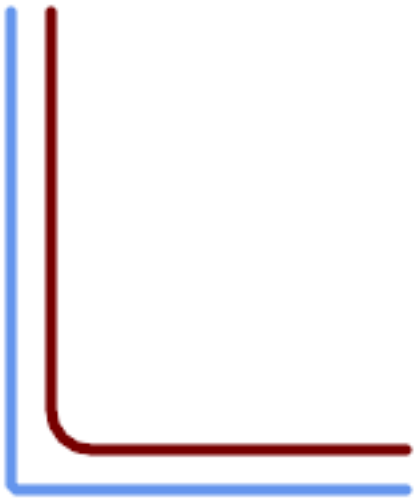
15, 'quad_segs=4 join=round' original line and its offset 15 units.

```
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,
    16 18,16 20,16 40,16 60,16 80,16 100,
    16 120,16 140,16 160,16 180,16 195)') ↩
    ,
    15, 'quad_segs=4 join=round'));

output

LINESTRING(164 1,18 1,12.2597485145237  ↩
    2.1418070123307,
    7.39339828220179 5.39339828220179,
    5.39339828220179 7.39339828220179,
    2.14180701233067 12.2597485145237,1  ↩
    18,1 195)
```



-15, 'quad_segs=4 join=round' original line and its offset -15 units

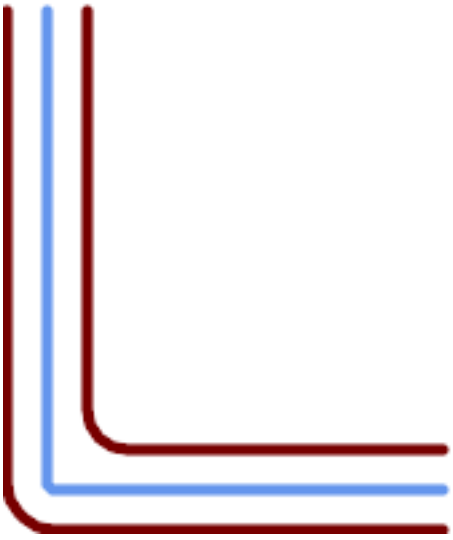
```
SELECT ST_AsText(ST_OffsetCurve(geom,
    -15, 'quad_segs=4 join=round')) As  ↩
    notsocurvy
    FROM ST_GeomFromText(
'LINESTRING(164 16,144 16,124 16,104  ↩
    16,84 16,64 16,
    44 16,24 16,20 16,18 16,17 17,
    16 18,16 20,16 40,16 60,16 80,16 100,
    16 120,16 140,16 160,16 180,16 195)') ↩
     As geom;

notsocurvy

LINESTRING(31 195,31 31,164 31)
```



double-offset para ficar mais curvo, note que o primeiro reverte a direção, então -30 + 15 = -15

```
SELECT ST_AsText(ST_OffsetCurve( ↩
    ST_OffsetCurve(geom,
    -30, 'quad_segs=4 join=round'), -15,  ↩
    'quad_segs=4 join=round')) As morecurvy
    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,
    16 18,16 20,16 40,16 60,16 80,16 100,
    16 120,16 140,16 160,16 180,16 195)') ↩
     As geom;
```

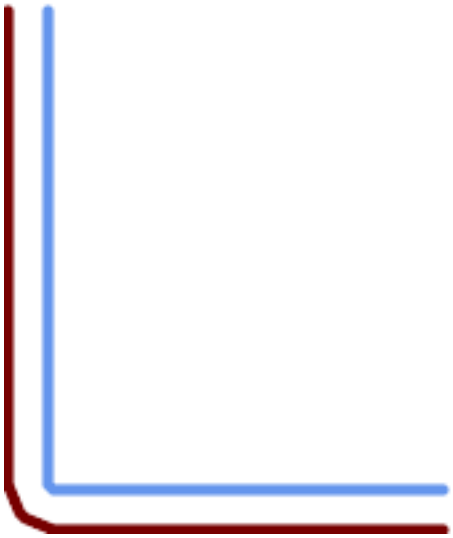
```
morecurvy

LINESTRING(164 31,46 31,40.2597485145236  ↩
    32.1418070123307,
35.3933982822018 35.3933982822018,
32.1418070123307 40.2597485145237,31  ↩
    46,31 195)
```



double-offset para ficar mais curvo, combinado com offset 15 para obter linhas paralelas. Coberto com o original.

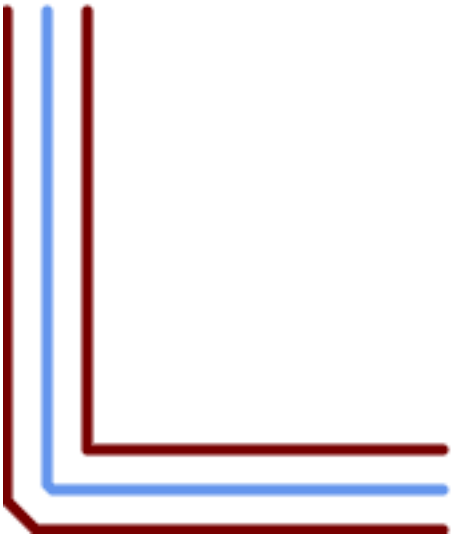
```
SELECT ST_AsText(ST_Collect(
    ST_OffsetCurve(geom, 15, 'quad_segs=4 ↩
     join=round'),
    ST_OffsetCurve(ST_OffsetCurve(geom,
    -30, 'quad_segs=4 join=round'), -15,  ↩
    'quad_segs=4 join=round')
    )
) As parallel_curves
    FROM ST_GeomFromText(
'LINESTRING(164 16,144 16,124 16,104  ↩
    16,84 16,64 16,
    44 16,24 16,20 16,18 16,17 17,
    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

MULTILINESTRING((164 1,18  ↩
    1,12.2597485145237 2.1418070123307,
7.39339828220179  ↩
    5.39339828220179,5.39339828220179 7.393398282201
2.14180701233067 12.2597485145237,1 18,1  ↩
    195),
(164 31,46 31,40.2597485145236  ↩
    32.1418070123307,35.3933982822018 35.39339828220
32.1418070123307 40.2597485145237,31  ↩
    46,31 195))
```



15, 'quad_segs=4 join=bevel' mostrado com a linha original

```
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,
    16 18,16 20,16 40,16 60,16 80,16 100,
    16 120,16 140,16 160,16 180,16 195)') ↩
    ,
        15, 'quad_segs=4 join=bevel'));

output

LINESTRING(164 1,18 1,7.39339828220179  ↩
    5.39339828220179,
    5.39339828220179 7.39339828220179,1  ↩
    18,1 195)
```



15,-15 collected, join=mitre mitre_limit=2.1

```
SELECT ST_AsText(ST_Collect(
    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')
    ) )
    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,
    16 18,16 20,16 40,16 60,16 80,16 100,
    16 120,16 140,16 160,16 180,16 195)') ↩
     As geom;

output

MULTILINESTRING((164 1,11.7867965644036  ↩
    1,1 11.7867965644036,1 195),
    (31 195,31 31,164 31))
```

Veja também.

ST_Buffer

7.14.18 ST_PointOnSurface

ST_PointOnSurface — Computes a point guaranteed to lie in a polygon, or on a geometry.

Synopsis

geometry **ST_PointOnSurface**(geometry g1);

Descrição

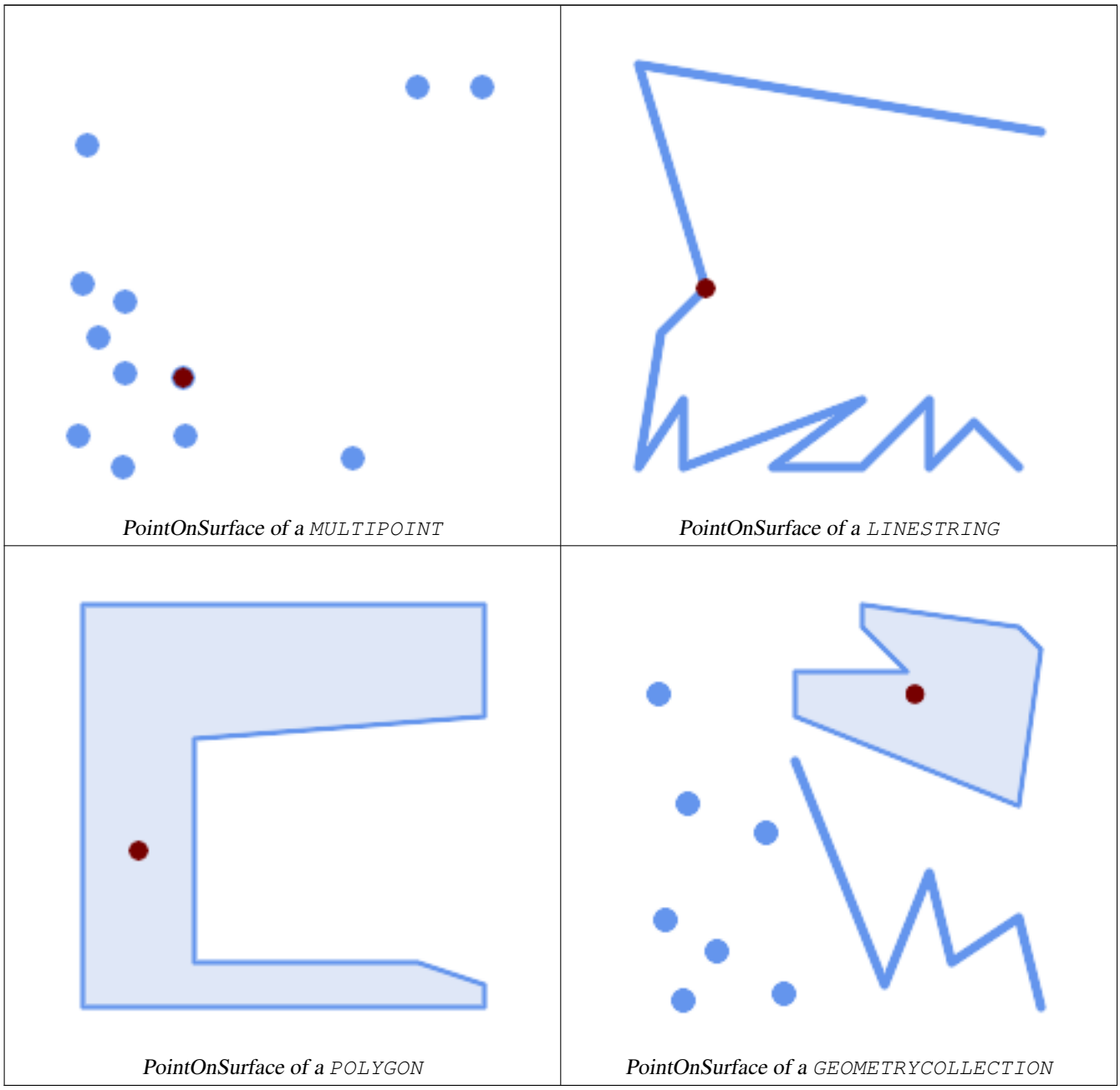
Returns a `POINT` which is guaranteed to lie in the interior of a surface (POLYGON, MULTIPOLYGON, and CURVED POLYGON). In PostGIS this function also works on line and point geometries.

This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s3.2.14.2 // s3.2.18.2

This method implements the SQL/MM specification. SQL-MM 3: 8.1.5, 9.5.6. The specifications define ST_PointOnSurface for surface geometries only. PostGIS extends the function to support all common geometry types. Other databases (Oracle, DB2, ArcSDE) seem to support this function only for surfaces. SQL Server 2008 supports all common geometry types.

This function supports 3d and will not drop the z-index.

Exemplos



PointOnSurface of a `MULTIPOINT`

PointOnSurface of a `LINESTRING`

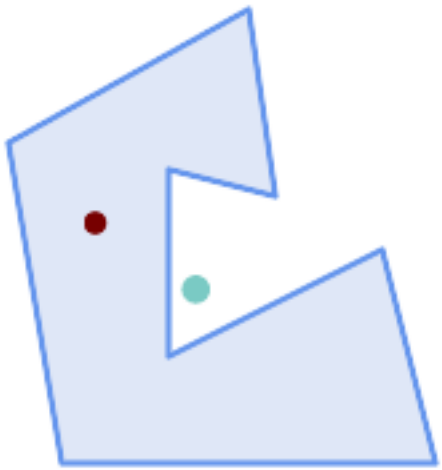
PointOnSurface of a `POLYGON`

PointOnSurface of a `GEOMETRYCOLLECTION`

```
SELECT ST_AsText(ST_PointOnSurface('POINT(0 5)'::geometry));
------------
 POINT(0 5)
```

```
SELECT ST_AsText(ST_PointOnSurface('LINESTRING(0 5, 0 10)'::geometry));
------------
 POINT(0 5)

SELECT ST_AsText(ST_PointOnSurface('POLYGON((0 0, 0 5, 5 5, 5 0, 0 0))'::geometry));
----------------
 POINT(2.5 2.5)

SELECT ST_AsEWKT(ST_PointOnSurface(ST_GeomFromEWKT('LINESTRING(0 5 1, 0 0 1, 0 10 2)')));
----------------
 POINT(0 0 1)
```

Example: The result of ST_PointOnSurface is guaranteed to lie within polygons, whereas the point computed by ST_Centroid may be outside.



Red: point on surface; Green: centroid

```
SELECT ST_AsText(ST_PointOnSurface(geom)) AS pt_on_surf,
       ST_AsText(ST_Centroid(geom)) AS centroid
    FROM (SELECT 'POLYGON ((130 120, 120 190, 30 140, 50 20, 190 20,
                      170 100, 90 60, 90 130, 130 120))'::geometry AS geom) AS t;

   pt_on_surf    |                  centroid
-----------------+-------------------------------------------
 POINT(62.5 110) | POINT(100.18264840182648 85.11415525114155)
```

Veja também.

ST_Centroid, ST_MaximumInscribedCircle

7.14.19 ST_Polygonize

ST_Polygonize — Computes a collection of polygons formed from the linework of a set of geometries.

Synopsis

geometry **ST_Polygonize**(geometry set geomfield);
geometry **ST_Polygonize**(geometry[] geom_array);

Descrição

Creates a GeometryCollection containing the polygons formed by the constituent linework of a set of geometries. Input linework must be correctly noded for this function to work properly.

Note
To ensure input is fully noded use ST_Node on the input geometry before polygonizing.

Note
GeometryCollections are often difficult to deal with with third party tools. Use ST_Dump to convert the polygonize result into separate polygons.

Desempenhado pelo módulo GEOS.

Disponibilidade: 1.0.0RC1

Exemplos: Poligonizando linestrings únicas

```
SELECT ST_AsEWKT(ST_Polygonize(geom_4269)) As geomtextrep
FROM (SELECT 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(geom_4269) As polycoll
    FROM (SELECT 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)
```

Veja também.

ST_Node, ST_Dump

7.14.20 ST_ReducePrecision

ST_ReducePrecision — Returns a valid geometry with points rounded to a grid tolerance.

Synopsis

geometry **ST_ReducePrecision**(geometry g, float8 gridsize);

Descrição

Returns a valid geometry with all points rounded to the provided grid tolerance, and features below the tolerance removed.

Unlike ST_SnapToGrid the returned geometry will be valid, with no ring self-intersections or collapsed components.

Precision reduction can be used to:

- match coordinate precision to the data accuracy
- reduce the number of coordinates needed to represent a geometry
- ensure valid geometry output to formats which use lower precision (e.g. text formats such as WKT, GeoJSON or KML when the number of output decimal places is limited).
- export valid geometry to systems which use lower or limited precision (e.g. SDE, Oracle tolerance value)

Availability: 3.1.0.

Requires GEOS >= 3.9.0.

Exemplos

```
SELECT ST_AsText(ST_ReducePrecision('POINT(1.412 19.323)', 0.1));
    st_astext
------------------
 POINT(1.4 19.3)

SELECT ST_AsText(ST_ReducePrecision('POINT(1.412 19.323)', 1.0));
  st_astext
-------------
 POINT(1 19)

SELECT ST_AsText(ST_ReducePrecision('POINT(1.412 19.323)', 10));
  st_astext
-------------
 POINT(0 20)
```

Precision reduction can reduce number of vertices

```
SELECT ST_AsText(ST_ReducePrecision('LINESTRING (10 10, 19.6 30.1, 20 30, 20.3 30, 40 40)', ↩
     1));
  st_astext
-------------
 LINESTRING (10 10, 20 30, 40 40)
```

Precision reduction splits polygons if needed to ensure validity

```
SELECT ST_AsText(ST_ReducePrecision('POLYGON ((10 10, 60 60.1, 70 30, 40 40, 50 10, 10 10)) ↩
    ', 10));
  st_astext
-------------
 MULTIPOLYGON (((60 60, 70 30, 40 40, 60 60)), ((40 40, 50 10, 10 10, 40 40)))
```

Veja também.

ST_SnapToGrid, ST_Simplify, ST_SimplifyVW

7.14.21 ST_SharedPaths

ST_SharedPaths — Retorna uma coleção contendo caminhos compartilhados pelas duas linestrings/multilinestrings de entrada.

Synopsis

geometry **ST_SharedPaths**(geometry lineal1, geometry lineal2);

Descrição

Retorna uma coleção contendo caminhos compartilhados pelas duas geometrias de entrada. Aquelas indo na mesma direção estão no primeiro elemento da coleção, aquelas indo na direção oposta estão no segundo elemento. Os caminhos por si mesmos são dados na direção da primeira geometria.

Desempenhado pelo módulo GEOS.

Disponibilidade: 2.0.0

Exemplos: Encontrando caminhos compartilhados

Uma multilinestring e uma linestring

O caminho compartilhado de multilinestring e linestring revestido com as geometrias originais.

```
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)))
```

Veja também.

ST_Dump, ST_GeometryN, ST_NumGeometries

7.14.22 ST_Simplify

ST_Simplify — Returns a simplified version of a geometry, using the Douglas-Peucker algorithm.

Synopsis

geometry **ST_Simplify**(geometry geomA, float tolerance);
geometry **ST_Simplify**(geometry geomA, float tolerance, boolean preserveCollapsed);

Descrição

Retorna uma versão da geometria dada com o algorítimo Douglas-Peucker. Só irá fazer algo com (multi)lines, (multi)polígonos e multipontos, mas você pode usar com qualquer tipo de geometria. Já que ocorre a simplificação em uma base objeto por objeto, você também pode alimentar uma GeometryCollection para esta função.

The "preserve collapsed" flag will retain objects that would otherwise be too small given the tolerance. For example, a 1m long line simplified with a 10m tolerance. If `preserveCollapsed` argument is specified as true, the line will not disappear. This flag is useful for rendering engines, to avoid having large numbers of very small objects disappear from a map leaving surprising gaps.

Note
Note que a geometria retornada pode perder sua simplicidade (veja ST_IsSimple)

Note
Note que a topologia pode não ser preservada e resultar em geometrias inválidas. Use (veja ST_SimplifyPreserveTopology) para preservar a topologia.

Disponibilidade: 1.2.2

Exemplos

Um círculo muito simplificado se torna um triângulo, médio um octágono,

```
SELECT ST_Npoints(geom) AS np_before,
       ST_NPoints(ST_Simplify(geom,0.1)) AS np01_notbadcircle,
       ST_NPoints(ST_Simplify(geom,0.5)) AS np05_notquitecircle,
       ST_NPoints(ST_Simplify(geom,1)) AS np1_octagon,
       ST_NPoints(ST_Simplify(geom,10)) AS np10_triangle,
       (ST_Simplify(geom,100) is null) AS  np100_geometrygoesaway
  FROM
    (SELECT ST_Buffer('POINT(1 3)', 10,12) As geom) AS foo;

 np_before | np01_notbadcircle | np05_notquitecircle | np1_octagon | np10_triangle |  ↩
     np100_geometrygoesaway
-----------+-------------------+---------------------+-------------+---------------+----------------

        49 |                33 |                  17 |           9 |             4 | t
```

Veja também.

ST_IsSimple, ST_SimplifyPreserveTopology, ST_SimplifyVW, Topology ST_Simplify

7.14.23 ST_SimplifyPreserveTopology

ST_SimplifyPreserveTopology — Returns a simplified and valid version of a geometry, using the Douglas-Peucker algorithm.

Synopsis

geometry **ST_SimplifyPreserveTopology**(geometry geomA, float tolerance);

Descrição

Retorna uma versão da geometria dada com o algorítimo Douglas-Peucker.Evitrá a criação de geometrias derivadas (polígonos, em particular) que sejam inválidas. Só irá fazer algo com (multi)lines, (multi)polígonos e multipontos, mas você pode usar com qualquer tipo de geometria. Já que ocorre a simplificação em uma base objeto por objeto, você também pode alimentar uma GeometryCollection para esta função.

Desempenhado pelo módulo GEOS.

Disponibilidade: 1.3.3

Exemplos

É o mesmo exemplo de Simplificar, mas vemos que Preserva a Topologia previne uma simplificação exagerada. O círculo pode se tornar, no máximo, um quadrado.

```
SELECT ST_Npoints(geom) As np_before, ST_NPoints(ST_SimplifyPreserveTopology(geom,0.1)) As  ↩
    np01_notbadcircle, ST_NPoints(ST_SimplifyPreserveTopology(geom,0.5)) As  ↩
    np05_notquitecircle,
ST_NPoints(ST_SimplifyPreserveTopology(geom,1)) As np1_octagon, ST_NPoints( ↩
    ST_SimplifyPreserveTopology(geom,10)) As np10_square,
ST_NPoints(ST_SimplifyPreserveTopology(geom,100)) As  np100_stillsquare
FROM (SELECT ST_Buffer('POINT(1 3)', 10,12) As geom) As foo;

--result--
 np_before | np01_notbadcircle | np05_notquitecircle | np1_octagon | np10_square |  ↩
     np100_stillsquare
-----------+-------------------+---------------------+-------------+---------------+----------------

        49 |                33 |                  17 |           9 |             5 |  ↩
                                5
```

Veja também.

ST_Simplify

7.14.24 ST_SimplifyPolygonHull

ST_SimplifyPolygonHull — Computes a simplifed topology-preserving outer or inner hull of a polygonal geometry.

Synopsis

geometry **ST_SimplifyPolygonHull**(geometry param_geom, float vertex_fraction, boolean is_outer = true);

Descrição

Computes a simplified topology-preserving outer or inner hull of a polygonal geometry. An outer hull completely covers the input geometry. An inner hull is completely covered by the input geometry. The result is a polygonal geometry formed by a subset of the input vertices. MultiPolygons and holes are handled and produce a result with the same structure as the input.

The reduction in vertex count is controlled by the `vertex_fraction` parameter, which is a number in the range 0 to 1. Lower values produce simpler results, with smaller vertex count and less concaveness. For both outer and inner hulls a vertex fraction of 1.0 produces the orginal geometry. For outer hulls a value of 0.0 produces the convex hull (for a single polygon); for inner hulls it produces a triangle.

The simplification process operates by progressively removing concave corners that contain the least amount of area, until the vertex count target is reached. It prevents edges from crossing, so the result is always a valid polygonal geometry.

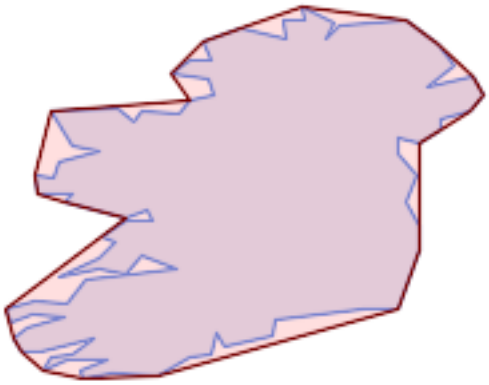
To get better results with geometries that contain relatively long line segments, it might be necessary to "segmentize" the input, as shown below.

Desempenhado pelo módulo GEOS.

Availability: 3.3.0.

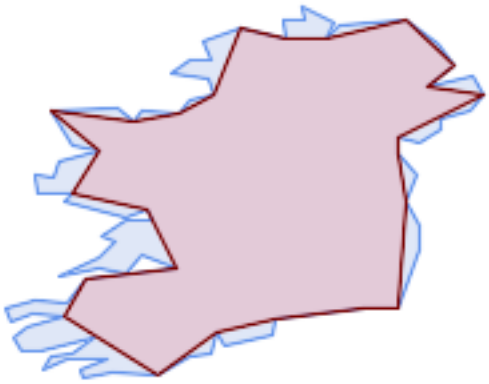
Requires GEOS >= 3.11.0.

Exemplos



Outer hull of a Polygon

```
SELECT ST_SimplifyPolygonHull(
  'POLYGON ((131 158, 136 163, 161 165, 173 156, 179 148, 169 140, 186 144, 190 137, 185  ↩
      131, 174 128, 174 124, 166 119, 158 121, 158 115, 165 107, 161 97, 166 88, 166 79, 158 ↩
       57, 145 57, 112 53, 111 47, 93 43, 90 48, 88 40, 80 39, 68 32, 51 33, 40 31, 39 34,  ↩
      49 38, 34 38, 25 34, 28 39, 36 40, 44 46, 24 41, 17 41, 14 46, 19 50, 33 54, 21 55, 13 ↩
       52, 11 57, 22 60, 34 59, 41 68, 75 72, 62 77, 56 70, 46 72, 31 69, 46 76, 52 82, 47  ↩
      84, 56 90, 66 90, 64 94, 56 91, 33 97, 36 100, 23 100, 22 107, 29 106, 31 112, 46 116, ↩
       36 118, 28 131, 53 132, 59 127, 62 131, 76 130, 80 135, 89 137, 87 143, 73 145, 80  ↩
      150, 88 150, 85 157, 99 162, 116 158, 115 165, 123 165, 122 170, 134 164, 131 158))',
    0.3);
```



Inner hull of a Polygon

```
SELECT ST_SimplifyPolygonHull(
  'POLYGON ((131 158, 136 163, 161 165, 173 156, 179 148, 169 140, 186 144, 190 137, 185 ↩
     131, 174 128, 174 124, 166 119, 158 121, 158 115, 165 107, 161 97, 166 88, 166 79, 158 ↩
      57, 145 57, 112 53, 111 47, 93 43, 90 48, 88 40, 80 39, 68 32, 51 33, 40 31, 39 34, ↩
     49 38, 34 38, 25 34, 28 39, 36 40, 44 46, 24 41, 17 41, 14 46, 19 50, 33 54, 21 55, 13 ↩
      52, 11 57, 22 60, 34 59, 41 68, 75 72, 62 77, 56 70, 46 72, 31 69, 46 76, 52 82, 47 ↩
     84, 56 90, 66 90, 64 94, 56 91, 33 97, 36 100, 23 100, 22 107, 29 106, 31 112, 46 116, ↩
      36 118, 28 131, 53 132, 59 127, 62 131, 76 130, 80 135, 89 137, 87 143, 73 145, 80 ↩
     150, 88 150, 85 157, 99 162, 116 158, 115 165, 123 165, 122 170, 134 164, 131 158))',
    0.3, false);
```

Outer hull simplification of a MultiPolygon, with segmentization

```
SELECT ST_SimplifyPolygonHull(
  ST_Segmentize(ST_Letters('xt'), 2.0),
    0.1);
```

Veja também.

ST_ConvexHull, ST_SimplifyVW, ST_ConcaveHull, ST_Segmentize

7.14.25 ST_SimplifyVW

ST_SimplifyVW — Returns a simplified version of a geometry, using the Visvalingam-Whyatt algorithm

Synopsis

geometry **ST_SimplifyVW**(geometry geomA, float tolerance);

Descrição

Retorna uma versão da geometria dada com o algorítimo Visvalingam-Whyatt. Só irá fazer algo com (multi)lines, (multi)polígonos e multipontos, mas você pode usar com qualquer tipo de geometria. Já que ocorre a simplificação em uma base objeto por objeto, você também pode alimentar uma GeometryCollection para esta função.

Note
Note que a geometria retornada pode perder sua simplicidade (veja ST_IsSimple)

Note
Note que a topologia pode não ser preservada e resultar em geometrias inválidas. Use (veja ST_SimplifyPreserveTopology) para preservar a topologia.

Note
Esta função lida com 3D e a terceira dimensão afetará o resultado.

Disponibilidade: 2.2.0

Exemplos

Uma LineString é simplificada com uma área mínima a ponto de 30.

```
select ST_AsText(ST_SimplifyVW(geom,30)) simplified
FROM (SELECT  'LINESTRING(5 2, 3 8, 6 20, 7 25, 10 10)'::geometry geom) As foo;
-result
 simplified
------------------------------
LINESTRING(5 2,7 25,10 10)
```

Veja também.

ST_SetEffectiveArea, ST_Simplify, ST_SimplifyPreserveTopology, Topology ST_Simplify

7.14.26 ST_SetEffectiveArea

ST_SetEffectiveArea — Sets the effective area for each vertex, using the Visvalingam-Whyatt algorithm.

Synopsis

geometry **ST_SetEffectiveArea**(geometry geomA, float threshold = 0, integer set_area = 1);

Descrição

Define a área eficaz para cada vértice, usando o algorítimo Visvalingam-Whyatt. A área efetiva é armazenada como o valor-M do vértice. Se o parâmetro opcional "limiar" for usado, uma geometria simplificada vai retornar, contendo somente vértices com uma área efetiva maior ou igual ao valor limiar.

Esta função pode ser usada para simplificação do lado do servidor quando uma limiar estiver especificada. Outra opção é usar um valor limiar de zero. Neste caso, a geometria completa retornará com áreas eficazes como valores-M, que podem ser usados pelo cliente para simplifica rapidamente.

Só irá fazer algo com (multi)lines, (multi)polígonos e multipontos, mas você pode usar com qualquer tipo de geometria. Já que ocorre a simplificação em uma base objeto por objeto, você também pode alimentar uma GeometryCollection para esta função.

Note
Note que a geometria retornada pode perder sua simplicidade (veja ST_IsSimple)

Note
Note que a topologia pode não ser preservada e resultar em geometrias inválidas. Use (veja ST_SimplifyPreserveTopology) para preservar a topologia.

Note
A geometria de saída perderá todas as informações prévias nos valores-M

Note
Esta função lida com 3D e a terceira dimensão afetará a área eficaz.

Disponibilidade: 2.2.0

Exemplos

Calculando a área eficaz de uma LineString. Devido ao uso de um valor limiar zero, todos os vértices na geometria de entrada são retornados.

```
select ST_AsText(ST_SetEffectiveArea(geom)) all_pts, ST_AsText(ST_SetEffectiveArea(geom,30) ↩
    ) thrshld_30
FROM (SELECT  'LINESTRING(5 2, 3 8, 6 20, 7 25, 10 10)'::geometry geom) As foo;
-result
 all_pts | thrshld_30
-----------+-------------------+
LINESTRING M (5 2 3.40282346638529e+38,3 8 29,6 20 1.5,7 25 49.5,10 10 3.40282346638529e ↩
   +38) | LINESTRING M (5 2 3.40282346638529e+38,7 25 49.5,10 10 3.40282346638529e+38)
```

Veja também.

ST_SimplifyVW

7.14.27 ST_TriangulatePolygon

ST_TriangulatePolygon — Computes the constrained Delaunay triangulation of polygons

Synopsis

geometry **ST_TriangulatePolygon**(geometry geom);

Descrição

Computes the constrained Delaunay triangulation of polygons. Holes and Multipolygons are supported.

The "constrained Delaunay triangulation" of a polygon is a set of triangles formed from the vertices of the polygon, and covering it exactly, with the maximum total interior angle over all possible triangulations. It provides the "best quality" triangulation of the polygon.

Availability: 3.3.0.

Requires GEOS >= 3.11.0.

Example

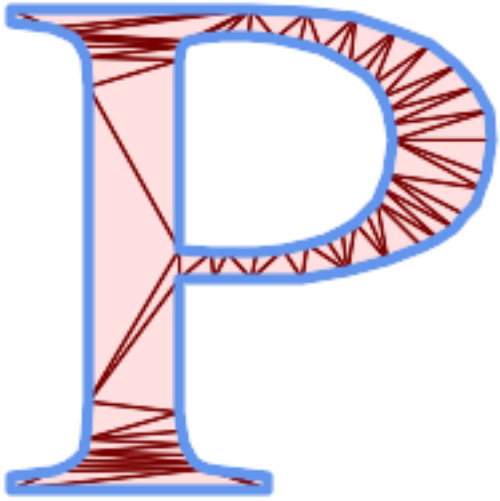
Triangulation of a square.

```
SELECT ST_AsText(
    ST_TriangulatePolygon('POLYGON((0 0, 0 1, 1 1, 1 0, 0 0))'));

                                st_astext
------------------------------------------------------------------------
 GEOMETRYCOLLECTION(POLYGON((0 0,0 1,1 1,0 0)),POLYGON((1 1,1 0,0 0,1 1)))
```

Example

Triangulation of the letter P.

```
SELECT ST_AsText(ST_TriangulatePolygon(
    'POLYGON ((26 17, 31 19, 34 21, 37 24, 38 29, 39 43, 39 161, 38 172, 36 176, 34 179, 30 ↩
         181, 25 183, 10 185, 10 190, 100 190, 121 189, 139 187, 154 182, 167 177, 177 169,  ↩
        184 161, 189 152, 190 141, 188 128, 186 123, 184 117, 180 113, 176 108, 170 104, 164 ↩
         101, 151 96, 136 92, 119 89, 100 89, 86 89, 73 89, 73 39, 74 32, 75 27, 77 23, 79  ↩
        20, 83 18, 89 17, 106 15, 106 10, 10 10, 10 15, 26 17), (152 147, 151 152, 149 157,  ↩
        146 162, 142 166, 137 169, 132 172, 126 175, 118 177, 109 179, 99 180, 89 180, 80  ↩
        179, 76 178, 74 176, 73 171, 73 100, 85 99, 91 99, 102 99, 112 100, 121 102, 128  ↩
        104, 134 107, 139 110, 143 114, 147 118, 149 123, 151 128, 153 141, 152 147))'
    ));
```



Polygon Triangulation

Veja também.

ST_DelaunayTriangles, ST_ConstrainedDelaunayTriangles, ST_Tesselate

7.14.28 ST_VoronoiLines

ST_VoronoiLines — Returns the boundaries of the Voronoi diagram of the vertices of a geometry.

Synopsis

geometry **ST_VoronoiLines**(geometry geom , float8 tolerance = 0.0 , geometry extend_to = NULL);

Descrição

Computes a two-dimensional Voronoi diagram from the vertices of the supplied geometry and returns the boundaries between cells in the diagram as a MultiLineString. Returns null if input geometry is null. Returns an empty geometry collection if the input geometry contains only one vertex. Returns an empty geometry collection if the `extend_to` envelope has zero area.

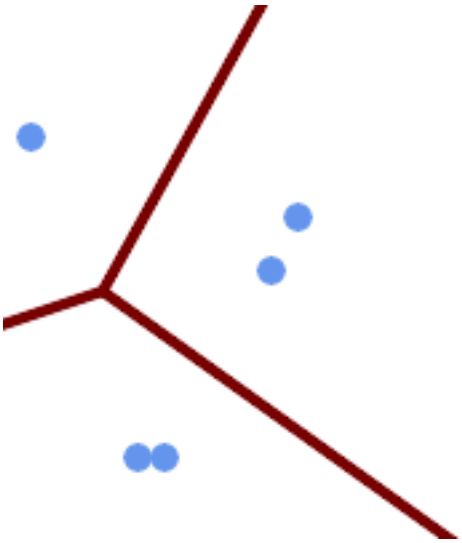
Parâmetros opcionais:

- `tolerance`: The distance within which vertices will be considered equivalent. Robustness of the algorithm can be improved by supplying a nonzero tolerance distance. (default = 0.0)
- `extend_to`: If present, the diagram is extended to cover the envelope of the supplied geometry, unless smaller than the default envelope (default = NULL, default envelope is the bounding box of the input expanded by about 50%).

Desempenhado pelo módulo GEOS.

Disponibilidade: 2.3.0

Exemplos



Voronoi diagram lines, with tolerance of 30 units

```
SELECT ST_VoronoiLines(
            'MULTIPOINT (50 30, 60 30, 100 100,10 150, 110 120)'::geometry,
            30) AS geom;
```

```
ST_AsText output
MULTILINESTRING((135.555555555556 270,36.8181818181818 92.2727272727273),(36.8181818181818 ↩
    92.2727272727273,-110 43.3333333333333),(230 -45.7142857142858,36.8181818181818 ↩
    92.2727272727273))
```

Veja também.

ST_DelaunayTriangles, ST_VoronoiPolygons

7.14.29 ST_VoronoiPolygons

ST_VoronoiPolygons — Returns the cells of the Voronoi diagram of the vertices of a geometry.

Synopsis

geometry **ST_VoronoiPolygons**(geometry geom , float8 tolerance = 0.0 , geometry extend_to = NULL);

Descrição

Computes a two-dimensional Voronoi diagram from the vertices of the supplied geometry. The result is a GEOMETRYCOLLECTION of POLYGONs that covers an envelope larger than the extent of the input vertices. Returns null if input geometry is null. Returns an empty geometry collection if the input geometry contains only one vertex. Returns an empty geometry collection if the `extend_to` envelope has zero area.

Parâmetros opcionais:

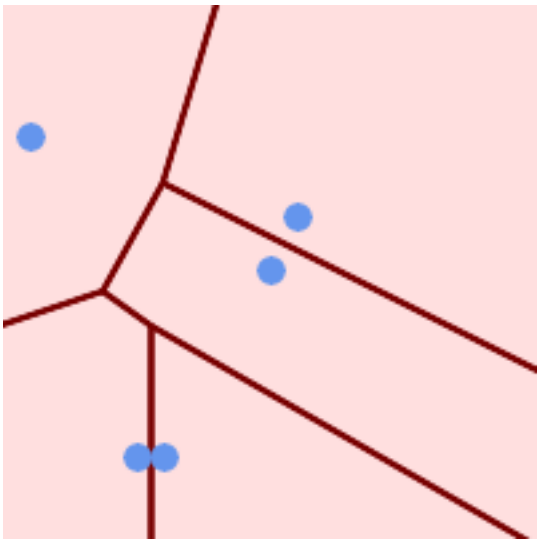
- `tolerance`: The distance within which vertices will be considered equivalent. Robustness of the algorithm can be improved by supplying a nonzero tolerance distance. (default = 0.0)

- `extend_to`: If present, the diagram is extended to cover the envelope of the supplied geometry, unless smaller than the default envelope (default = NULL, default envelope is the bounding box of the input expanded by about 50%).

Desempenhado pelo módulo GEOS.

Disponibilidade: 2.3.0

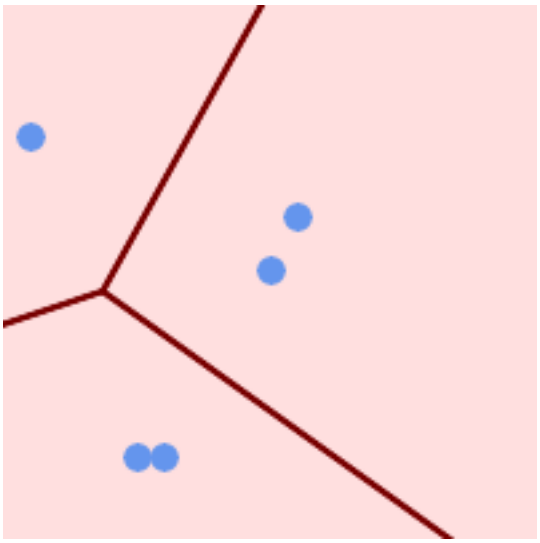
Exemplos



Points overlaid on top of Voronoi diagram

```
SELECT ST_VoronoiPolygons(
                'MULTIPOINT (50 30, 60 30, 100 100,10 150, 110 120)'::geometry
            ) AS geom;
```

```
ST_AsText output
GEOMETRYCOLLECTION(POLYGON((-110 43.3333333333333,-110 270,100.5 270,59.3478260869565  ↩
    132.826086956522,36.8181818181818 92.2727272727273,-110 43.3333333333333)),
POLYGON((55 -90,-110 -90,-110 43.3333333333333,36.8181818181818 92.2727272727273,55  ↩
    79.2857142857143,55 -90)),
POLYGON((230 47.5,230 -20.7142857142857,55 79.2857142857143,36.8181818181818  ↩
    92.2727272727273,59.3478260869565 132.826086956522,230 47.5)),POLYGON((230  ↩
    -20.7142857142857,230 -90,55 -90,55 79.2857142857143,230 -20.7142857142857)),
POLYGON((100.5 270,230 270,230 47.5,59.3478260869565 132.826086956522,100.5 270)))
```



Voronoi diagram, with tolerance of 30 units

```
SELECT ST_VoronoiPolygons(
            'MULTIPOINT (50 30, 60 30, 100 100,10 150, 110 120)'::geometry,
            30) AS geom;
```

```
ST_AsText output
GEOMETRYCOLLECTION(POLYGON((-110 43.3333333333333,-110 270,100.5 270,59.3478260869565  ↩
    132.826086956522,36.8181818181818 92.2727272727273,-110 43.3333333333333)),
POLYGON((230 47.5,230 -45.7142857142858,36.8181818181818 92.2727272727273,59.3478260869565  ↩
    132.826086956522,230 47.5)),POLYGON((230 -45.7142857142858,230 -90,-110 -90,-110  ↩
    43.3333333333333,36.8181818181818 92.2727272727273,230 -45.7142857142858)),
POLYGON((100.5 270,230 270,230 47.5,59.3478260869565 132.826086956522,100.5 270)))
```

Veja também.

ST_DelaunayTriangles, ST_VoronoiLines

7.15 Coverages

7.15.1 ST_CoverageInvalidEdges

ST_CoverageInvalidEdges — Window function that finds locations where polygons fail to form a valid coverage.

Synopsis

geometry **ST_CoverageInvalidEdges**(geometry winset geom, float8 tolerance = 0);

Description

A window function which checks if the polygons in the window partition form a valid polygonal coverage. It returns linear indicators showing the location of invalid edges (if any) in each polygon.

A set of valid polygons is a valid coverage if the following conditions hold:

- **Non-overlapping** - polygons do not overlap (their interiors do not intersect)

- **Edge-Matched** - vertices along shared edges are identical

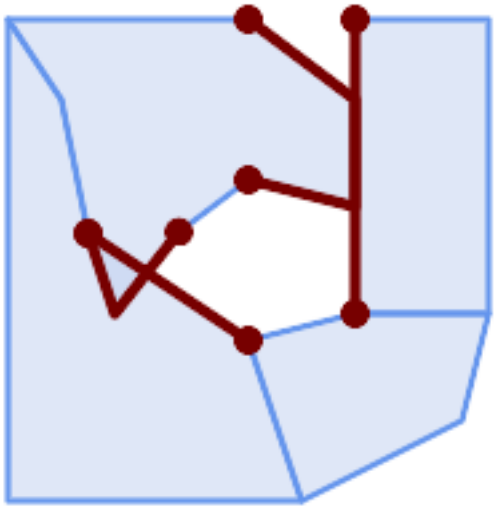
As a window function a value is returned for every input polygon. For polygons which violate one or more of the validity conditions the return value is a MULTILINESTRING containing the problematic edges. Coverage-valid polygons return the value NULL. Non-polygonal or empty geometries also produce NULL values.

The conditions allow a valid coverage to contain holes (gaps between polygons), as long as the surrounding polygons are edge-matched. However, very narrow gaps are often undesirable. If the *tolerance* parameter is specified with a non-zero distance, edges forming narrower gaps will also be returned as invalid.

The polygons being checked for coverage validity must also be valid geometries. This can be checked with ST_IsValid.

Availability: 3.4.0 - requires GEOS >= 3.12.0

Examples



Invalid edges caused by overlap and non-matching vertices

```
WITH coverage(id, geom) AS (VALUES
  (1, 'POLYGON ((10 190, 30 160, 40 110, 100 70, 120 10, 10 10, 10 190))'::geometry),
  (2, 'POLYGON ((100 190, 10 190, 30 160, 40 110, 50 80, 74 110.5, 100 130, 140 120, 140  ↩
     160, 100 190))'::geometry),
  (3, 'POLYGON ((140 190, 190 190, 190 80, 140 80, 140 190))'::geometry),
  (4, 'POLYGON ((180 40, 120 10, 100 70, 140 80, 190 80, 180 40))'::geometry)
)
SELECT id, ST_AsText(ST_CoverageInvalidEdges(geom) OVER ())
  FROM coverage;

 id |               st_astext
----+----------------------------------------
  1 | LINESTRING (40 110, 100 70)
  2 | MULTILINESTRING ((100 130, 140 120, 140 160, 100 190), (40 110, 50 80, 74 110.5))
  3 | LINESTRING (140 80, 140 190)
  3 | null
```

```
-- Test entire table for coverage validity
SELECT true = ALL (
    SELECT ST_CoverageInvalidEdges(geom) OVER () IS NULL
    FROM coverage
    );
```

See Also

ST_IsValid, ST_CoverageUnion, ST_CoverageSimplify

7.15.2 ST_CoverageSimplify

ST_CoverageSimplify — Window function that simplifies the edges of a polygonal coverage.

Synopsis

geometry **ST_CoverageSimplify**(geometry winset geom, float8 tolerance, boolean simplifyBoundary = true);

Description

A window function which simplifies the edges of polygons in a polygonal coverage. The simplification preserves the coverage topology. This means the simplified output polygons are consisent along shared edges, and still form a valid coverage.

The simplification uses a variant of the Visvalingam–Whyatt algorithm. The `tolerance` parameter has units of distance, and is roughly equal to the square root of triangular areas to be simplified.

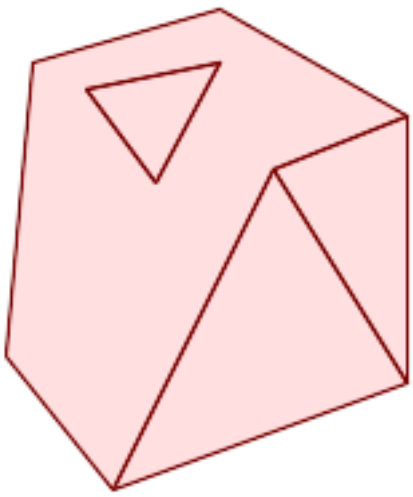
To simplify only the "internal" edges of the coverage (those that are shared by two polygons) set the `simplifyBoundary` parameter to false.

Note
If the input is not a valid coverage there may be unexpected artifacts in the output (such as boundary intersections, or separated boundaries which appeared to be shared). Use ST_CoverageInvalidEdges to determine if a coverage is valid.

Availability: 3.4.0 - requires GEOS >= 3.12.0

Examples

Input coverage



Simplified coverage

```
WITH coverage(id, geom) AS (VALUES
  (1, 'POLYGON ((160 150, 110 130, 90 100, 90 70, 60 60, 50 10, 30 30, 40 50, 25 40, 10 60, ↩
       30 100, 30 120, 20 170, 60 180, 90 190, 130 180, 130 160, 160 150), (40 160, 50 140,  ↩
      66 125, 60 100, 80 140, 90 170, 60 160, 40 160))'::geometry),
  (2, 'POLYGON ((40 160, 60 160, 90 170, 80 140, 60 100, 66 125, 50 140, 40 160))':: ↩
      geometry),
  (3, 'POLYGON ((110 130, 160 50, 140 50, 120 33, 90 30, 50 10, 60 60, 90 70, 90 100, 110  ↩
      130))'::geometry),
  (4, 'POLYGON ((160 150, 150 120, 160 90, 160 50, 110 130, 160 150))'::geometry)
)
SELECT id, ST_AsText(ST_CoverageSimplify(geom, 30) OVER ())
  FROM coverage;

 id |               st_astext
----+---------------------------------------
  1 | POLYGON ((160 150, 110 130, 50 10, 10 60, 20 170, 90 190, 160 150), (40 160, 66 125,  ↩
      90 170, 40 160))
  2 | POLYGON ((40 160, 66 125, 90 170, 40 160))
  3 | POLYGON ((110 130, 160 50, 50 10, 110 130))
  3 | POLYGON ((160 150, 160 50, 110 130, 160 150))
```

See Also

ST_CoverageInvalidEdges

7.15.3 ST_CoverageUnion

ST_CoverageUnion — Computes the union of a set of polygons forming a coverage by removing shared edges.

Synopsis

geometry **ST_CoverageUnion**(geometry set geom);

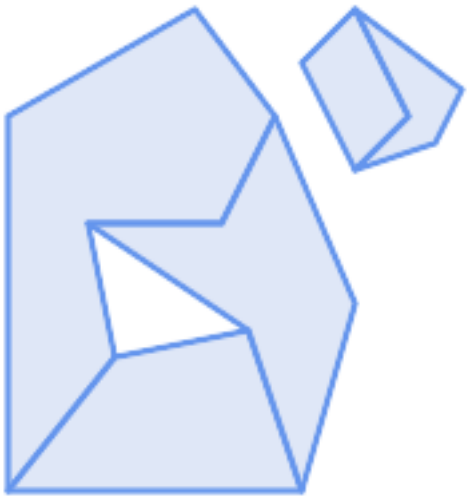
Description

An aggregate function which unions a set of polygons forming a polygonal coverage. The result is a polygonal geometry covering the same area as the coverage. This function produces the same result as ST_Union, but uses the coverage structure to compute the union much faster.

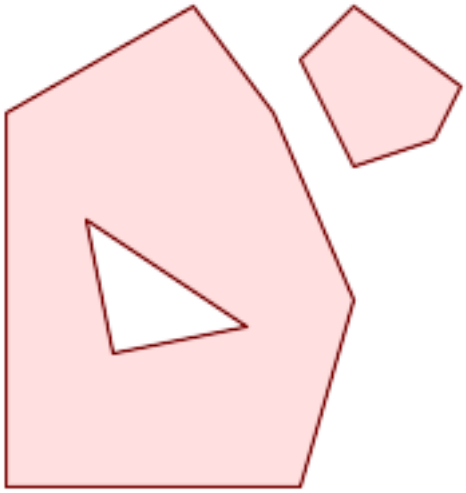
Note
If the input is not a valid coverage there may be unexpected artifacts in the output (such as unmerged or overlapping polygons). Use ST_CoverageInvalidEdges to determine if a coverage is valid.

Availability: 3.4.0 - requires GEOS >= 3.8.0

Examples



Input coverage



Union result

```
WITH coverage(id, geom) AS (VALUES
  (1, 'POLYGON ((10 10, 10 150, 80 190, 110 150, 90 110, 40 110, 50 60, 10 10))'::geometry) ↩
      ,
  (2, 'POLYGON ((120 10, 10 10, 50 60, 100 70, 120 10))'::geometry),
  (3, 'POLYGON ((140 80, 120 10, 100 70, 40 110, 90 110, 110 150, 140 80))'::geometry),
  (4, 'POLYGON ((140 190, 120 170, 140 130, 160 150, 140 190))'::geometry),
  (5, 'POLYGON ((180 160, 170 140, 140 130, 160 150, 140 190, 180 160))'::geometry)
)
SELECT ST_AsText(ST_CoverageUnion(geom))
  FROM coverage;

--------------------------------------
MULTIPOLYGON (((10 150, 80 190, 110 150, 140 80, 120 10, 10 10, 10 150), (50 60, 100 70, 40 ↩
     110, 50 60)), ((120 170, 140 190, 180 160, 170 140, 140 130, 120 170)))
```

See Also

ST_CoverageInvalidEdges, ST_Union

7.16 Affine Transformations

7.16.1 ST_Affine

ST_Affine — Apply a 3D affine transformation to a geometry.

Synopsis

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);

Description

Applies a 3D affine transformation to the geometry to do things like translate, rotate, scale in one step.

Version 1: The call

```
ST_Affine(geom, a, b, c, d, e, f, g, h, i, xoff, yoff, zoff)
```

represents the transformation matrix

```
/ a  b  c  xoff \
| d  e  f  yoff |
| g  h  i  zoff |
\ 0  0  0     1 /
```

and the vertices are transformed as follows:

```
x' = a*x + b*y + c*z + xoff
y' = d*x + e*y + f*z + yoff
z' = g*x + h*y + i*z + zoff
```

All of the translate / scale functions below are expressed via such an affine transformation.

Version 2: Applies a 2d affine transformation to the geometry. The call

```
ST_Affine(geom, a, b, d, e, xoff, yoff)
```

represents the transformation matrix

```
/  a  b  0  xoff  \       /  a  b  xoff  \
|  d  e  0  yoff  | rsp.  |  d  e  yoff  |
|  0  0  1     0  |       \  0  0     1  /
\  0  0  0     1  /
```

and the vertices are transformed as follows:

```
x' = a*x + b*y + xoff
y' = d*x + e*y + yoff
z' = z
```

This method is a subcase of the 3D method above.

Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced.

Availability: 1.1.2. Name changed from Affine to ST_Affine in 1.2.2

Note

Prior to 1.3.4, this function crashes if used with geometries that contain CURVES. This is fixed in 1.3.4+

 This function supports Polyhedral surfaces.

 This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

 This function supports 3d and will not drop the z-index.

 This method supports Circular Strings and Curves.

Examples

```
--Rotate a 3d line 180 degrees about the z axis.  Note this is long-hand for doing  ↩
    ST_Rotate();
 SELECT ST_AsEWKT(ST_Affine(geom,  cos(pi()), -sin(pi()), 0,  sin(pi()), cos(pi()), 0,  0,  ↩
     0, 1,  0, 0, 0)) As using_affine,
          ST_AsEWKT(ST_Rotate(geom, pi())) As using_rotate
        FROM (SELECT ST_GeomFromEWKT('LINESTRING(1 2 3, 1 4 3)') As 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(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 geom) As foo;
           st_asewkt
--------------------------------
 LINESTRING(-1 -2 -3,-1 -4 -3)
(1 row)
```

See Also

ST_Rotate, ST_Scale, ST_Translate, ST_TransScale

7.16.2 ST_Rotate

ST_Rotate — Rotates a geometry about an origin point.

Synopsis

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);

Description

Rotates geometry rotRadians counter-clockwise about the origin point. The rotation origin can be specified either as a POINT geometry, or as x and y coordinates. If the origin is not specified, the geometry is rotated about POINT(0 0).

Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced.

Enhanced: 2.0.0 additional parameters for specifying the origin of rotation were added.

Availability: 1.1.2. Name changed from Rotate to ST_Rotate in 1.2.2

 This function supports 3d and will not drop the z-index.

 This method supports Circular Strings and Curves.

 This function supports Polyhedral surfaces.

 This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

Examples

```
--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)
(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)
```

See Also

ST_Affine, ST_RotateX, ST_RotateY, ST_RotateZ

7.16.3 ST_RotateX

ST_RotateX — Rotates a geometry about the X axis.

Synopsis

geometry **ST_RotateX**(geometry geomA, float rotRadians);

Description

Rotates a geometry geomA - rotRadians about the X axis.

Note

`ST_RotateX(geomA, rotRadians)` is short-hand for `ST_Affine(geomA, 1, 0, 0, 0, cos(rotRadians), -sin(rotRadians), 0, sin(rotRadians), cos(rotRadians), 0, 0, 0).`

Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced.

Availability: 1.1.2. Name changed from RotateX to ST_RotateX in 1.2.2


This function supports Polyhedral surfaces.


This function supports 3d and will not drop the z-index.


This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

Examples

```
--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)
```

See Also

ST_Affine, ST_RotateY, ST_RotateZ

7.16.4 ST_RotateY

ST_RotateY — Rotates a geometry about the Y axis.

Synopsis

geometry **ST_RotateY**(geometry geomA, float rotRadians);

Description

Rotates a geometry geomA - rotRadians about the y axis.

Note
`ST_RotateY(geomA, rotRadians)` is short-hand for `ST_Affine(geomA, cos(rotRadians), 0, sin(rotRadians), 0, 1, 0, -sin(rotRadians), 0, cos(rotRadians), 0, 0, 0)`.

Availability: 1.1.2. Name changed from RotateY to ST_RotateY in 1.2.2

Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced.


This function supports Polyhedral surfaces.


This function supports 3d and will not drop the z-index.


This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

Examples

```
--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)
```

See Also

ST_Affine, ST_RotateX, ST_RotateZ

7.16.5 ST_RotateZ

ST_RotateZ — Rotates a geometry about the Z axis.

Synopsis

geometry **ST_RotateZ**(geometry geomA, float rotRadians);

Description

Rotates a geometry geomA - rotRadians about the Z axis.

Note
This is a synonym for ST_Rotate

Note
`ST_RotateZ(geomA, rotRadians)` is short-hand for `SELECT ST_Affine(geomA, cos(rotRadians), -sin(rotRadians), 0, sin(rotRadians), cos(rotRadians), 0, 0, 0, 1, 0, 0, 0).`

Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced.

Availability: 1.1.2. Name changed from RotateZ to ST_RotateZ in 1.2.2

Note
Prior to 1.3.4, this function crashes if used with geometries that contain CURVES. This is fixed in 1.3.4+


This function supports 3d and will not drop the z-index.


This method supports Circular Strings and Curves.


This function supports Polyhedral surfaces.


This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

Examples

```
--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(geom, pi()/2))
FROM (SELECT ST_LineToCurve(ST_Buffer(ST_GeomFromText('POINT(234 567)'), 3)) As geom) As  ↩
    foo;
```

```
----------------------------------------------------------------------------------------------------------------

 CURVEPOLYGON(CIRCULARSTRING(-567 237,-564.87867965644 236.12132034356,-564  ↩
     234,-569.12132034356 231.87867965644,-567 237))
```

See Also

ST_Affine, ST_RotateX, ST_RotateY

7.16.6 ST_Scale

ST_Scale — Scales a geometry by given factors.

Synopsis

geometry **ST_Scale**(geometry geomA, float XFactor, float YFactor, float ZFactor);
geometry **ST_Scale**(geometry geomA, float XFactor, float YFactor);
geometry **ST_Scale**(geometry geom, geometry factor);
geometry **ST_Scale**(geometry geom, geometry factor, geometry origin);

Description

Scales the geometry to a new size by multiplying the ordinates with the corresponding factor parameters.

The version taking a geometry as the `factor` parameter allows passing a 2d, 3dm, 3dz or 4d point to set scaling factor for all supported dimensions. Missing dimensions in the `factor` point are equivalent to no scaling the corresponding dimension.

The three-geometry variant allows a "false origin" for the scaling to be passed in. This allows "scaling in place", for example using the centroid of the geometry as the false origin. Without a false origin, scaling takes place relative to the actual origin, so all coordinates are just multipled by the scale factor.

Note
Prior to 1.3.4, this function crashes if used with geometries that contain CURVES. This is fixed in 1.3.4+

Availability: 1.1.0.

Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced.

Enhanced: 2.2.0 support for scaling all dimension (`factor` parameter) was introduced.

Enhanced: 2.5.0 support for scaling relative to a local origin (`origin` parameter) was introduced.

 This function supports Polyhedral surfaces.

 This function supports 3d and will not drop the z-index.

 This method supports Circular Strings and Curves.

 This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

 This function supports M coordinates.

Examples

```
--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));
                         st_asewkt
---------------------------------
 LINESTRING(0.5 1.5 3,0.5 0.75 1)

--Version 3: Scale X Y Z M
 SELECT ST_AsEWKT(ST_Scale(ST_GeomFromEWKT('LINESTRING(1 2 3 4, 1 1 1 1)'),
   ST_MakePoint(0.5, 0.75, 2, -1)));
                                st_asewkt
-----------------------------------------
 LINESTRING(0.5 1.5 6 -4,0.5 0.75 2 -1)

--Version 4: Scale X Y using false origin
SELECT ST_AsText(ST_Scale('LINESTRING(1 1, 2 2)', 'POINT(2 2)', 'POINT(1 1)'::geometry));
      st_astext
---------------------
 LINESTRING(1 1,3 3)
```

See Also

ST_Affine, ST_TransScale

7.16.7 ST_Translate

ST_Translate — Translates a geometry by given offsets.

Synopsis

geometry **ST_Translate**(geometry g1, float deltax, float deltay);
geometry **ST_Translate**(geometry g1, float deltax, float deltay, float deltaz);

Description

Returns a new geometry whose coordinates are translated delta x,delta y,delta z units. Units are based on the units defined in spatial reference (SRID) for this geometry.

Note
Prior to 1.3.4, this function crashes if used with geometries that contain CURVES. This is fixed in 1.3.4+

Availability: 1.2.2

 This function supports 3d and will not drop the z-index.

 This method supports Circular Strings and Curves.

Examples

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));


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

 GEOMETRYCOLLECTION(CURVEPOLYGON(CIRCULARSTRING(5 5,4.12 2.878,2 2,-0.121 7.1213,7 9,9 11,5 ↩
     5)),POINT(2 5))
```

See Also

ST_Affine, ST_AsText, ST_GeomFromText

7.16.8 ST_TransScale

ST_TransScale — Translates and scales a geometry by given offsets and factors.

Synopsis

geometry **ST_TransScale**(geometry geomA, float deltaX, float deltaY, float XFactor, float YFactor);

Description

Translates the geometry using the deltaX and deltaY args, then scales it using the XFactor, YFactor args, working in 2D only.

Note

`ST_TransScale(geomA, deltaX, deltaY, XFactor, YFactor)` is short-hand for `ST_Affine(geomA, XFactor, 0, 0, 0, YFactor, 0, 0, 0, 1, deltaX*XFactor, deltaY*YFactor, 0)`.

Note
Prior to 1.3.4, this function crashes if used with geometries that contain CURVES. This is fixed in 1.3.4+

Availability: 1.1.0.


This function supports 3d and will not drop the z-index.


This method supports Circular Strings and Curves.

Examples

```
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));


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

 CURVEPOLYGON(CIRCULARSTRING(714 2276,711.363961030679 2267.51471862576,705  ↩
     2264,698.636038969321 2284.48528137424,714 2276))
```

See Also

ST_Affine, ST_Translate

7.17 Clustering Functions

7.17.1 ST_ClusterDBSCAN

ST_ClusterDBSCAN — Window function that returns a cluster id for each input geometry using the DBSCAN algorithm.

Synopsis

integer **ST_ClusterDBSCAN**(geometry winset geom, float8 eps, integer minpoints);

Description

Returns cluster number for each input geometry, based on a 2D implementation of the Density-based spatial clustering of applications with noise (DBSCAN) algorithm. Unlike ST_ClusterKMeans, it does not require the number of clusters to be specified, but instead uses the desired distance (`eps`) and density (`minpoints`) parameters to construct each cluster.

An input geometry will be added to a cluster if it is either:

- A "core" geometry, that is within `eps` distance of at least `minpoints` input geometries (including itself) or
- A "border" geometry, that is within `eps` distance of a core geometry.

Note that border geometries may be within `eps` distance of core geometries in more than one cluster; in this case, either assignment would be correct, and the border geometry will be arbitrarily asssigned to one of the available clusters. In these cases, it is possible for a correct cluster to be generated with fewer than `minpoints` geometries. When assignment of a border geometry is ambiguous, repeated calls to ST_ClusterDBSCAN will produce identical results if an ORDER BY clause is included in the window definition, but cluster assignments may differ from other implementations of the same algorithm.

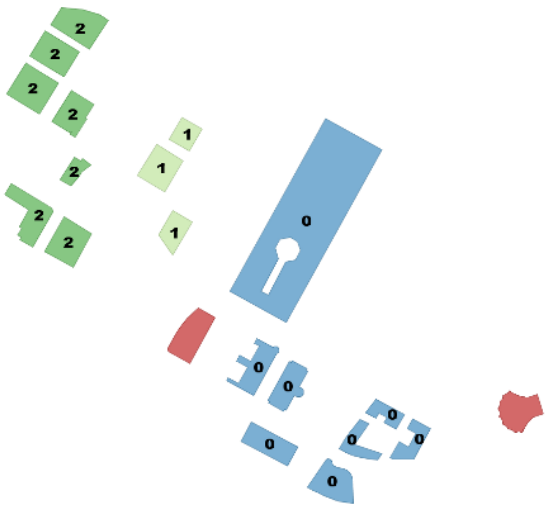
Note
Input geometries that do not meet the criteria to join any other cluster will be assigned a cluster number of NULL.

Availability: 2.3.0

 This method supports Circular Strings and Curves.

Examples

Assigning a cluster number to each polygon within 50 meters of each other. Require at least 2 polygons per cluster



within 50 meters at least 2 per cluster. singletons have NULL for cid

```
SELECT name, ST_ClusterDBSCAN(geom, eps  ↩
    := 50, minpoints := 2) over () AS cid
FROM boston_polys
WHERE name
> '' AND building
> ''
        AND ST_DWithin(geom,
        ST_Transform(
            ST_GeomFromText('POINT ↩
    (-71.04054 42.35141)', 4326), 26986),
          500);
```

```
               name                   |  ↩
    bucket
--------------------------------------+-------- ↩

 Manulife Tower                       |  ↩
         0
 Park Lane Seaport I                  |  ↩
         0
 Park Lane Seaport II                 |  ↩
         0
 Renaissance Boston Waterfront Hotel  |  ↩
         0
 Seaport Boston Hotel                 |  ↩
         0
 Seaport Hotel & World Trade Center   |  ↩
         0
 Waterside Place                      |  ↩
         0
 World Trade Center East              |  ↩
         0
 100 Northern Avenue                  |  ↩
         1
 100 Pier 4                           |  ↩
         1
 The Institute of Contemporary Art    |  ↩
         1
 101 Seaport                          |  ↩
         2
 District Hall                        |  ↩
         2
 One Marina Park Drive                |  ↩
         2
 Twenty Two Liberty                   |  ↩
         2
 Vertex                               |  ↩
         2
 Vertex                               |  ↩
         2
 Watermark Seaport                    |  ↩
         2
 Blue Hills Bank Pavilion             |    ↩
    NULL
 World Trade Center West              |    ↩
    NULL
(20 rows)
```

Combining parcels with the same cluster number into a single geometry. This uses named argument calling

```
SELECT cid, ST_Collect(geom) AS cluster_geom, array_agg(parcel_id) AS ids_in_cluster FROM (
    SELECT parcel_id, ST_ClusterDBSCAN(geom, eps := 0.5, minpoints := 5) over () AS cid,  ↩
        geom
    FROM parcels) sq
GROUP BY cid;
```

See Also

ST_DWithin, ST_ClusterKMeans, ST_ClusterIntersecting, ST_ClusterWithin ST_ClusterIntersectingWin

7.17.2 ST_ClusterIntersectingWin

ST_ClusterIntersectingWin — Window function that returns a cluster id for each input geometry, clustering input geometries into connected sets.

Synopsis

integer **ST_ClusterIntersectingWin**(geometry winset geom);

Description

ST_ClusterIntersectingWin is a windowing function that builds inter-connecting clusters of geometries that intersect. It is possible to traverse all geometries in a cluster without leaving the cluster. The return value is the cluster number that the geometry argument participates in, or null for null inputs.

Availability: 3.4.0

Examples

```
WITH testdata AS (
  SELECT id, geom::geometry FROM (
  VALUES  (1, 'LINESTRING (0 0, 1 1)'),
          (2, 'LINESTRING (5 5, 4 4)'),
          (3, 'LINESTRING (6 6, 7 7)'),
          (4, 'LINESTRING (0 0, -1 -1)'),
          (5, 'POLYGON ((0 0, 4 0, 4 4, 0 4, 0 0))')) AS t(id, geom)
)
SELECT id,
  ST_AsText(geom),
  ST_ClusterIntersectingWin(geom) OVER () AS cluster
FROM testdata;

 id |           st_astext            | cluster
----+--------------------------------+---------
  1 | LINESTRING(0 0,1 1)            |       0
  2 | LINESTRING(5 5,4 4)            |       0
  3 | LINESTRING(6 6,7 7)            |       1
  4 | LINESTRING(0 0,-1 -1)          |       0
  5 | POLYGON((0 0,4 0,4 4,0 4,0 0)) |       0
```

See Also

ST_ClusterDBSCAN, ST_ClusterKMeans, ST_ClusterWithinWin ST_ClusterWithin ST_ClusterIntersecting

7.17.3 ST_ClusterIntersecting

ST_ClusterIntersecting — Aggregate function that clusters input geometries into connected sets.

Synopsis

geometry[] **ST_ClusterIntersecting**(geometry set g);

Description

ST_ClusterIntersecting is an aggregate function that returns an array of GeometryCollections, where each GeometryCollection represents an interconnected set of geometries.

Availability: 2.2.0

Examples

```
WITH testdata AS
  (SELECT unnest(ARRAY['LINESTRING (0 0, 1 1)'::geometry,
           'LINESTRING (5 5, 4 4)'::geometry,
           'LINESTRING (6 6, 7 7)'::geometry,
           'LINESTRING (0 0, -1 -1)'::geometry,
           'POLYGON ((0 0, 4 0, 4 4, 0 4, 0 0))'::geometry]) AS geom)

SELECT ST_AsText(unnest(ST_ClusterIntersecting(geom))) FROM testdata;

--result

st_astext
---------
GEOMETRYCOLLECTION(LINESTRING(0 0,1 1),LINESTRING(5 5,4 4),LINESTRING(0 0,-1 -1),POLYGON((0 ↩
     0,4 0,4 4,0 4,0 0)))
GEOMETRYCOLLECTION(LINESTRING(6 6,7 7))
```

See Also

ST_ClusterDBSCAN, ST_ClusterKMeans, ST_ClusterWithinWin ST_ClusterWithin

7.17.4 ST_ClusterKMeans

ST_ClusterKMeans — Window function that returns a cluster id for each input geometry using the K-means algorithm.

Synopsis

integer **ST_ClusterKMeans**(geometry winset geom, integer number_of_clusters, float max_radius);

Description

Returns K-means cluster number for each input geometry. The distance used for clustering is the distance between the centroids for 2D geometries, and distance between bounding box centers for 3D geometries. For POINT inputs, M coordinate will be treated as weight of input and has to be larger than 0.

`max_radius`, if set, will cause ST_ClusterKMeans to generate more clusters than `k` ensuring that no cluster in output has radius larger than `max_radius`. This is useful in reachability analysis.

Enhanced: 3.2.0 Support for `max_radius`

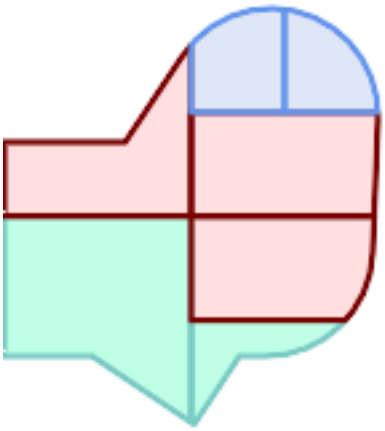
Enhanced: 3.1.0 Support for 3D geometries and weights

Availability: 2.3.0

Examples

Generate dummy set of parcels for examples:

```
CREATE TABLE parcels AS
SELECT lpad((row_number() over())::text,3,'0') As parcel_id, geom,
('{residential, commercial}'::text[])[1 + mod(row_number()OVER(),2)] As type
FROM
    ST_Subdivide(ST_Buffer('SRID=3857;LINESTRING(40 100, 98 100, 100 150, 60 90)'::geometry ↩
        ,
    40, 'endcap=square'),12) As geom;
```



Parcels color-coded by cluster number (cid)

```
SELECT ST_ClusterKMeans(geom, 3) OVER() AS cid, parcel_id, geom
    FROM parcels;
```

```
cid | parcel_id |   geom
-----+-----------+---------------
   0 | 001       | 0103000000...
   0 | 002       | 0103000000...
   1 | 003       | 0103000000...
   0 | 004       | 0103000000...
   1 | 005       | 0103000000...
   2 | 006       | 0103000000...
   2 | 007       | 0103000000...
```

Partitioning parcel clusters by type:

```
SELECT ST_ClusterKMeans(geom, 3) over (PARTITION BY type) AS cid, parcel_id, type
    FROM parcels;
```

```
cid | parcel_id |    type
-----+-----------+-------------
   1 | 005       | commercial
   1 | 003       | commercial
   2 | 007       | commercial
   0 | 001       | commercial
   1 | 004       | residential
   0 | 002       | residential
   2 | 006       | residential
```

Example: Clustering a preaggregated planetary-scale data population dataset using 3D clusering and weighting. Identify at least 20 regions based on Kontur Population Data that do not span more than 3000 km from their center:

```
create table kontur_population_3000km_clusters as
select
    geom,
    ST_ClusterKMeans(
        ST_Force4D(
            ST_Transform(ST_Force3D(geom), 4978), -- cluster in 3D XYZ CRS
            mvalue := population -- set clustering to be weighed by population
        ),
        20,                      -- aim to generate at least 20 clusters
        max_radius := 3000000    -- but generate more to make each under 3000 km radius
    ) over () as cid
from
    kontur_population;
```

World population clustered to above specs produces 46 clusters. Clusters are centered at well-populated regions (New York, Moscow). Greenland is one cluster. There are island clusters that span across the antimeridian. Cluster edges follow Earth's curvature.

See Also

ST_ClusterDBSCAN, ST_ClusterIntersectingWin, ST_ClusterWithinWin, ST_ClusterIntersecting, ST_ClusterWithin, ST_Subdivide, ST_Force3D, ST_Force4D,

7.17.5 ST_ClusterWithin

ST_ClusterWithin — Aggregate function that clusters input geometries by separation distance.

Synopsis

geometry[] **ST_ClusterWithin**(geometry set g, float8 distance);

Description

ST_ClusterWithin is an aggregate function that returns an array of GeometryCollections, where each GeometryCollection represents a set of geometries separated by no more than the specified distance. (Distances are Cartesian distances in the units of the SRID.)

Availability: 2.2.0

This method supports Circular Strings and Curves.

Examples

```
WITH testdata AS
  (SELECT unnest(ARRAY['LINESTRING (0 0, 1 1)'::geometry,
                       'LINESTRING (5 5, 4 4)'::geometry,
                       'LINESTRING (6 6, 7 7)'::geometry,
                       'LINESTRING (0 0, -1 -1)'::geometry,
                       'POLYGON ((0 0, 4 0, 4 4, 0 4, 0 0))'::geometry]) AS geom)

SELECT ST_AsText(unnest(ST_ClusterWithin(geom, 1.4))) FROM testdata;

--result

st_astext
---------
GEOMETRYCOLLECTION(LINESTRING(0 0,1 1),LINESTRING(5 5,4 4),LINESTRING(0 0,-1 -1),POLYGON((0 ↩
     0,4 0,4 4,0 4,0 0)))
GEOMETRYCOLLECTION(LINESTRING(6 6,7 7))
```

See Also

ST_ClusterDBSCAN, ST_ClusterKMeans, ST_ClusterIntersectingWin, ST_ClusterWithinWin, ST_ClusterIntersecting

7.17.6 ST_ClusterWithinWin

ST_ClusterWithinWin — Window function that returns a cluster id for each input geometry, clustering using separation distance.

Synopsis

integer **ST_ClusterWithinWin**(geometry winset geom, float8 distance);

Description

ST_ClusterWithinWin is a window function that returns an integer for each input geometry, where the integer the cluster number the geometry is a member of. Clusters represent a set of input geometries separated by no more than the specified distance. (Distances are Cartesian distances in the units of the SRID.)

ST_ClusterWithinWin is equivalent to running ST_ClusterDBSCAN with a `minpoints` of zero.

Availability: 3.4.0

 This method supports Circular Strings and Curves.

Examples

```
WITH testdata AS (
  SELECT id, geom::geometry FROM (
  VALUES  (1, 'LINESTRING (0 0, 1 1)'),
          (2, 'LINESTRING (5 5, 4 4)'),
          (3, 'LINESTRING (6 6, 7 7)'),
          (4, 'LINESTRING (0 0, -1 -1)'),
          (5, 'POLYGON ((0 0, 4 0, 4 4, 0 4, 0 0))')) AS t(id, geom)
)
SELECT id,
  ST_AsText(geom),
  ST_ClusterWithinWin(geom, 1.4) OVER () AS cluster
```

```
FROM testdata;


 id |           st_astext             | cluster
----+---------------------------------+---------
  1 | LINESTRING(0 0,1 1)             |       0
  2 | LINESTRING(5 5,4 4)             |       0
  3 | LINESTRING(6 6,7 7)             |       1
  4 | LINESTRING(0 0,-1 -1)           |       0
  5 | POLYGON((0 0,4 0,4 4,0 4,0 0)) |       0
```

See Also

ST_ClusterDBSCAN, ST_ClusterKMeans, ST_ClusterIntersectingWin, ST_ClusterWithin, ST_ClusterIntersecting

7.18 Bounding Box Functions

7.18.1 Box2D

Box2D — Returns a BOX2D representing the 2D extent of a geometry.

Synopsis

box2d **Box2D**(geometry geom);

Description

Returns a box2d representing the 2D extent of the geometry.

Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced.

 This method supports Circular Strings and Curves.

 This function supports Polyhedral surfaces.

 This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

Examples

```
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)
```

See Also

Box3D, ST_GeomFromText

7.18.2 Box3D

Box3D — Returns a BOX3D representing the 3D extent of a geometry.

Synopsis

box3d **Box3D**(geometry geom);

Description

Returns a box3d representing the 3D extent of the geometry.

Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced.

 This method supports Circular Strings and Curves.

 This function supports Polyhedral surfaces.

 This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

 This function supports 3d and will not drop the z-index.

Examples

```
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)
```

See Also

Box2D, ST_GeomFromEWKT

7.18.3 ST_EstimatedExtent

ST_EstimatedExtent — Returns the estimated extent of a spatial table.

Synopsis

box2d **ST_EstimatedExtent**(text schema_name, text table_name, text geocolumn_name, boolean parent_only);
box2d **ST_EstimatedExtent**(text schema_name, text table_name, text geocolumn_name);
box2d **ST_EstimatedExtent**(text table_name, text geocolumn_name);

Description

Returns the estimated extent of a spatial table as a box2d. The current schema is used if not specified. The estimated extent is taken from the geometry column's statistics. This is usually much faster than computing the exact extent of the table using ST_Extent or ST_3DExtent.

The default behavior is to also use statistics collected from child tables (tables with INHERITS) if available. If `parent_only` is set to TRUE, only statistics for the given table are used and child tables are ignored.

For PostgreSQL >= 8.0.0 statistics are gathered by VACUUM ANALYZE and the result extent will be about 95% of the actual one. For PostgreSQL < 8.0.0 statistics are gathered by running `update_geometry_stats()` and the result extent is exact.

Note
In the absence of statistics (empty table or no ANALYZE called) this function returns NULL. Prior to version 1.5.4 an exception was thrown instead.

Availability: 1.0.0

Changed: 2.1.0. Up to 2.0.x this was called ST_Estimated_Extent.

 This method supports Circular Strings and Curves.

Examples

```
SELECT ST_EstimatedExtent('ny', 'edges', 'geom');
--result--
BOX(-8877653 4912316,-8010225.5 5589284)

SELECT ST_EstimatedExtent('feature_poly', 'geom');
--result--
BOX(-124.659652709961 24.6830825805664,-67.7798080444336 49.0012092590332)
```

See Also

ST_Extent, ST_3DExtent

7.18.4 ST_Expand

ST_Expand — Returns a bounding box expanded from another bounding box or a geometry.

Synopsis

geometry **ST_Expand**(geometry geom, float units_to_expand);
geometry **ST_Expand**(geometry geom, float dx, float dy, float dz=0, float dm=0);
box2d **ST_Expand**(box2d box, float units_to_expand);
box2d **ST_Expand**(box2d box, float dx, float dy);
box3d **ST_Expand**(box3d box, float units_to_expand);
box3d **ST_Expand**(box3d box, float dx, float dy, float dz=0);

Description

Returns a bounding box expanded from the bounding box of the input, either by specifying a single distance with which the box should be expanded on both axes, or by specifying an expansion distance for each axis. Uses double-precision. Can be used for distance queries, or to add a bounding box filter to a query to take advantage of a spatial index.

In addition to the version of ST_Expand accepting and returning a geometry, variants are provided that accept and return box2d and box3d data types.

Distances are in the units of the spatial reference system of the input.

ST_Expand is similar to ST_Buffer, except while buffering expands a geometry in all directions, ST_Expand expands the bounding box along each axis.

Note
Pre version 1.3, ST_Expand was used in conjunction with ST_Distance to do indexable distance queries. For example, `geom && ST_Expand('POINT(10 20)', 10) AND ST_Distance(geom, 'POINT(10 20)') < 10`. This has been replaced by the simpler and more efficient ST_DWithin function.

Availability: 1.5.0 behavior changed to output double precision instead of float4 coordinates.

Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced.

Enhanced: 2.3.0 support was added to expand a box by different amounts in different dimensions.


This function supports Polyhedral surfaces.


This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

Examples

Note
Examples below use US National Atlas Equal Area (SRID=2163) which is a meter projection

```
--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))
```

See Also

ST_Buffer, ST_DWithin, ST_SRID

7.18.5 ST_Extent

ST_Extent — Aggregate function that returns the bounding box of geometries.

Synopsis

box2d **ST_Extent**(geometry set geomfield);

Description

An aggregate function that returns a box2d bounding box that bounds a set of geometries.

The bounding box coordinates are in the spatial reference system of the input geometries.

ST_Extent is similar in concept to Oracle Spatial/Locator's SDO_AGGR_MBR.

Note
ST_Extent returns boxes with only X and Y ordinates even with 3D geometries. To return XYZ ordinates use ST_3DExtent.

Note
The returned `box3d` value does not include a SRID. Use ST_SetSRID to convert it into a geometry with SRID metadata. The SRID is the same as the input geometries.

Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced.

This function supports Polyhedral surfaces.

This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

Examples

Note
Examples below use Massachusetts State Plane ft (SRID=2249)

```
SELECT ST_Extent(geom) as bextent FROM sometable;
                                          st_bextent
------------------------------------
BOX(739651.875 2908247.25,794875.8125 2970042.75)


--Return extent of each category of geometries
SELECT ST_Extent(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(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))
```

See Also

ST_EstimatedExtent, ST_3DExtent, ST_SetSRID

7.18.6 ST_3DExtent

ST_3DExtent — Aggregate function that returns the 3D bounding box of geometries.

Synopsis

box3d **ST_3DExtent**(geometry set geomfield);

Description

An aggregate function that returns a box3d (includes Z ordinate) bounding box that bounds a set of geometries.

The bounding box coordinates are in the spatial reference system of the input geometries.

Note
The returned `box3d` value does not include a SRID. Use ST_SetSRID to convert it into a geometry with SRID metadata. The SRID is the same as the input geometries.

Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced.

Changed: 2.0.0 In prior versions this used to be called ST_Extent3D

 This function supports 3d and will not drop the z-index.

 This method supports Circular Strings and Curves.

 This function supports Polyhedral surfaces.

 This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

Examples

```
SELECT ST_3DExtent(foo.geom) As b3extent
FROM (SELECT ST_MakePoint(x,y,z) As 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 1 0,3 2 2)

--Get the extent of various elevated circular strings
SELECT ST_3DExtent(foo.geom) As b3extent
FROM (SELECT ST_Translate(ST_Force_3DZ(ST_LineToCurve(ST_Buffer(ST_Point(x,y),1))),0,0,z)  ↩
    As 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)
```

See Also

ST_Extent, ST_Force3DZ, ST_SetSRID

7.18.7 ST_MakeBox2D

ST_MakeBox2D — Creates a BOX2D defined by two 2D point geometries.

Synopsis

box2d **ST_MakeBox2D**(geometry pointLowLeft, geometry pointUpRight);

Description

Creates a box2d defined by two Point geometries. This is useful for doing range queries.

Examples

```
--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, geom
FROM features
WHERE geom && ST_SetSRID(ST_MakeBox2D(ST_Point(-989502.1875, 528439.5625),
        ST_Point(-987121.375 ,529933.1875)),2163)
```

See Also

ST_Point, ST_SetSRID, ST_SRID

7.18.8 ST_3DMakeBox

ST_3DMakeBox — Creates a BOX3D defined by two 3D point geometries.

Synopsis

box3d **ST_3DMakeBox**(geometry point3DLowLeftBottom, geometry point3DUpRightTop);

Description

Creates a box3d defined by two 3D Point geometries.

 This function supports 3D and will not drop the z-index.

Changed: 2.0.0 In prior versions this used to be called ST_MakeBox3D

Examples

```
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)
```

See Also

ST_MakePoint, ST_SetSRID, ST_SRID

7.18.9 ST_XMax

ST_XMax — Returns the X maxima of a 2D or 3D bounding box or a geometry.

Synopsis

float **ST_XMax**(box3d aGeomorBox2DorBox3D);

Description

Returns the X maxima of a 2D or 3D bounding box or a geometry.

Note
Although this function is only defined for box3d, it also works for box2d and geometry values due to automatic casting. However, it will not accept a geometry or box2d text representation, since those do not auto-cast.

 This function supports 3d and will not drop the z-index.

 This method supports Circular Strings and Curves.

Examples

```
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 auto-cast the string representation to ↩
    a BOX3D
SELECT ST_XMax('LINESTRING(1 3, 5 6)');

--ERROR:  BOX3D parser - doesn't start with BOX3D(

SELECT ST_XMax(ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227 ↩
    150406 3)'));
st_xmax
--------
220288.248780547
```

See Also

ST_XMin, ST_YMax, ST_YMin, ST_ZMax, ST_ZMin

7.18.10 ST_XMin

ST_XMin — Returns the X minima of a 2D or 3D bounding box or a geometry.

Synopsis

float **ST_XMin**(box3d aGeomorBox2DorBox3D);

Description

Returns the X minima of a 2D or 3D bounding box or a geometry.

Note
Although this function is only defined for box3d, it also works for box2d and geometry values due to automatic casting. However it will not accept a geometry or box2d text representation, since those do not auto-cast.


This function supports 3d and will not drop the z-index.


This method supports Circular Strings and Curves.

Examples

```
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 auto-cast the string representation to ↩
    a BOX3D
SELECT ST_XMin('LINESTRING(1 3, 5 6)');

--ERROR:  BOX3D parser - doesn't start with BOX3D(

SELECT ST_XMin(ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227 ↩
    150406 3)'));
st_xmin
--------
220186.995121892
```

See Also

ST_XMax, ST_YMax, ST_YMin, ST_ZMax, ST_ZMin

7.18.11 ST_YMax

ST_YMax — Returns the Y maxima of a 2D or 3D bounding box or a geometry.

Synopsis

float **ST_YMax**(box3d aGeomorBox2DorBox3D);

Description

Returns the Y maxima of a 2D or 3D bounding box or a geometry.

Note
Although this function is only defined for box3d, it also works for box2d and geometry values due to automatic casting. However it will not accept a geometry or box2d text representation, since those do not auto-cast.


This function supports 3d and will not drop the z-index.


This method supports Circular Strings and Curves.

Examples

```
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 auto-cast the string representation to ↩
    a BOX3D
SELECT ST_YMax('LINESTRING(1 3, 5 6)');

--ERROR:  BOX3D parser - doesn't start with BOX3D(

SELECT ST_YMax(ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227  ↩
    150406 3)'));
st_ymax
--------
150506.126829327
```

See Also

ST_XMin, ST_XMax, ST_YMin, ST_ZMax, ST_ZMin

7.18.12 ST_YMin

ST_YMin — Returns the Y minima of a 2D or 3D bounding box or a geometry.

Synopsis

float **ST_YMin**(box3d aGeomorBox2DorBox3D);

Description

Returns the Y minima of a 2D or 3D bounding box or a geometry.

Note
Although this function is only defined for box3d, it also works for box2d and geometry values due to automatic casting. However it will not accept a geometry or box2d text representation, since those do not auto-cast.

 This function supports 3d and will not drop the z-index.

 This method supports Circular Strings and Curves.

Examples

```
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 auto-cast the string representation to ↩
    a BOX3D
SELECT ST_YMin('LINESTRING(1 3, 5 6)');

--ERROR:  BOX3D parser - doesn't start with BOX3D(

SELECT ST_YMin(ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227  ↩
    150406 3)'));
st_ymin
--------
150406
```

See Also

ST_GeomFromEWKT, ST_XMin, ST_XMax, ST_YMax, ST_ZMax, ST_ZMin

7.18.13 ST_ZMax

ST_ZMax — Returns the Z maxima of a 2D or 3D bounding box or a geometry.

Synopsis

float **ST_ZMax**(box3d aGeomorBox2DorBox3D);

Description

Returns the Z maxima of a 2D or 3D bounding box or a geometry.

Note
Although this function is only defined for box3d, it also works for box2d and geometry values due to automatic casting. However it will not accept a geometry or box2d text representation, since those do not auto-cast.

 This function supports 3d and will not drop the z-index.

 This method supports Circular Strings and Curves.

Examples

```
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 auto-cast the string representation to ↩
    a BOX3D
SELECT ST_ZMax('LINESTRING(1 3 4, 5 6 7)');

--ERROR:  BOX3D parser - doesn't start with BOX3D(

SELECT ST_ZMax(ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227  ↩
    150406 3)'));
st_zmax
--------
3
```

See Also

ST_GeomFromEWKT, ST_XMin, ST_XMax, ST_YMax, ST_YMin, ST_ZMax

7.18.14 ST_ZMin

ST_ZMin — Returns the Z minima of a 2D or 3D bounding box or a geometry.

Synopsis

float **ST_ZMin**(box3d aGeomorBox2DorBox3D);

Description

Returns the Z minima of a 2D or 3D bounding box or a geometry.

Note
Although this function is only defined for box3d, it also works for box2d and geometry values due to automatic casting. However it will not accept a geometry or box2d text representation, since those do not auto-cast.

 This function supports 3d and will not drop the z-index.

 This method supports Circular Strings and Curves.

Examples

```
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

SELECT ST_ZMin('BOX3D(-3 2 1, 3 4 1)' );
st_zmin
-------
1
--Observe THIS DOES NOT WORK because it will try to auto-cast the string representation to ↩
    a BOX3D
SELECT ST_ZMin('LINESTRING(1 3 4, 5 6 7)');

--ERROR:  BOX3D parser - doesn't start with BOX3D(

SELECT ST_ZMin(ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227  ↩
    150406 3)'));
st_zmin
--------
1
```

See Also

ST_GeomFromEWKT, ST_GeomFromText, ST_XMin, ST_XMax, ST_YMax, ST_YMin, ST_ZMax

7.19 Referência linear

7.19.1 ST_LineInterpolatePoint

ST_LineInterpolatePoint — Returns a point interpolated along a line at a fractional location.

Synopsis

geometry **ST_LineInterpolatePoint**(geometry a_linestring, float8 a_fraction);
geography **ST_LineInterpolatePoint**(geography a_linestring, float8 a_fraction, boolean use_spheroid = true);

Descrição

Retorna um ponto interpolar com uma linha. Primeiro argumento deve ser uma LINESTRING. Segundo argumento é um float8 entre 0 e 1 representando fração do comprimento total da linestring do ponto tem que ser localizado.

Veja ST_LineLocatePoint para computar a linha de localização mais perto de um ponto.

Note
This function computes points in 2D and then interpolates values for Z and M, while ST_LineInterpolatePoint computes points in 3D and only interpolates the M value.

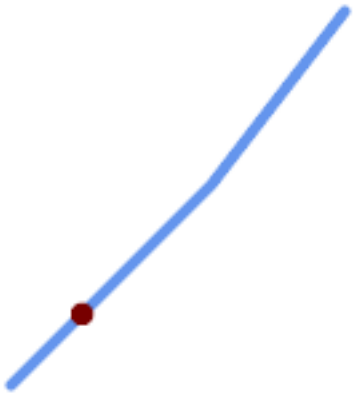
Note
Desde a liberação 1.1.1 essa função também interpola valores M e Z (quando presentes), enquanto as liberações anteriores configura eles para 0.0.

Disponibilidade: 0.8.2, Suporte a Z e M adicionado em 1.1.1

Alterações: 2.1.0 para 2.0.x foi chamada ST_Line_Interpolate_Point.

This function supports 3d and will not drop the z-index.

Exemplos



Uma linestring com o ponto interpolado em uma posição de 20% (0.20)

```
-- The point 20% along a line

SELECT ST_AsEWKT(  ST_LineInterpolatePoint(
        'LINESTRING(25 50, 100 125, 150 190)',
        0.2 ));
----------------
 POINT(51.5974135047432 76.5974135047432)
```

The mid-point of a 3D line:

```
SELECT ST_AsEWKT(  ST_LineInterpolatePoint('
        LINESTRING(1 2 3, 4 5 6, 6 7 8)',
        0.5 ));
--------------------
 POINT(3.5 4.5 5.5)
```

The closest point on a line to a point:

```
SELECT ST_AsText( ST_LineInterpolatePoint( line.geom,
                      ST_LineLocatePoint( line.geom, 'POINT(4 3)')))
FROM (SELECT ST_GeomFromText('LINESTRING(1 2, 4 5, 6 7)') As geom) AS line;
------------
 POINT(3 4)
```

Veja Também

ST_LineInterpolatePoints, ST_LineInterpolatePoint, ST_LineMerge

7.19.2 ST_LineInterpolatePoint

ST_LineInterpolatePoint — Returns a point interpolated along a 3D line at a fractional location.

Synopsis

geometria **ST_LineInterpolatePoint**(geometria a_linestring, float8 a_fraction);

Descrição

Retorna um ponto interpolar com uma linha. Primeiro argumento deve ser uma LINESTRING. Segundo argumento é um float8 entre 0 e 1 representando fração do comprimento total da linestring do ponto tem que ser localizado.

Note
ST_LineInterpolatePoint computes points in 2D and then interpolates the values for Z and M, while this function computes points in 3D and only interpolates the M value.

Disponibilidade: 2.0.0

 This function supports 3d and will not drop the z-index.

Exemplos

Return point 20% along 3D line

```
SELECT ST_AsText(
    ST_3DLineInterpolatePoint('LINESTRING(25 50 70, 100 125 90, 150 190 200)',
        0.20));

   st_asetext
----------------
 POINT Z (59.0675892910822 84.0675892910822 79.0846904776219)
```

Veja Também

ST_LineInterpolatePoint, ST_LineInterpolatePoint, ST_LineMerge

7.19.3 ST_LineInterpolatePoints

ST_LineInterpolatePoints — Returns points interpolated along a line at a fractional interval.

Synopsis

geometry **ST_LineInterpolatePoints**(geometry a_linestring, float8 a_fraction, boolean repeat);
geography **ST_LineInterpolatePoints**(geography a_linestring, float8 a_fraction, boolean use_spheroid = true, boolean repeat = true);

Descrição

Returns one or more points interpolated along a line at a fractional interval. The first argument must be a LINESTRING. The second argument is a float8 between 0 and 1 representing the spacing between the points as a fraction of line length. If the third argument is false, at most one point will be constructed (which is equivalent to ST_LineInterpolatePoint.)

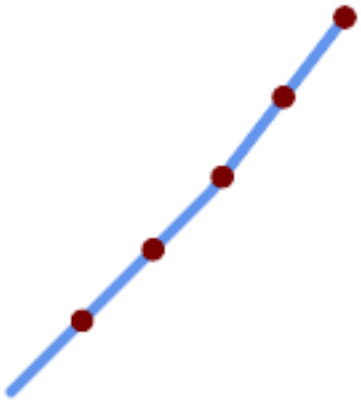
If the result has zero or one points, it is returned as a POINT. If it has two or more points, it is returned as a MULTIPOINT.

Availability: 2.5.0

This function supports 3d and will not drop the z-index.

This function supports M coordinates.

Exemplos



A LineString with points interpolated every 20%

```
--Return points each 20% along a 2D line
SELECT ST_AsText(ST_LineInterpolatePoints('LINESTRING(25 50, 100 125, 150 190)', 0.20))
----------------
 MULTIPOINT((51.5974135047432 76.5974135047432),(78.1948270094864 103.194827009486) ↩
    ,(104.132163186446 130.37181214238),(127.066081593223 160.18590607119),(150 190))
```

Veja Também

ST_LineInterpolatePoint, ST_LineLocatePoint

7.19.4 ST_LineLocatePoint

ST_LineLocatePoint — Returns the fractional location of the closest point on a line to a point.

Synopsis

float8 **ST_LineLocatePoint**(geometry a_linestring, geometry a_point);
float8 **ST_LineLocatePoint**(geography a_linestring, geography a_point, boolean use_spheroid = true);

Descrição

Retorna um flutuador entre 0 e 1 representando a localização do ponto mai próximo na linestring do ponto dado, como uma fração de uma 2d line de comprimento total.

Você pode usar a localização retornada para extrair um ponto (ST_LineInterpolatePoint) ou uma substring (ST_LineSubstring).

Isso é útil para aproximar números de endereços

Disponibilidade: 1.1.0

Alterações: 2.1.0 para 2.0.x foi chamada ST_Line_Locate_Point.

Exemplos

```
--Rough approximation of finding the street number of a point along the street
--Note 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_Point(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)
```

Veja Também

ST_DWithin, ST_Length2D, ST_LineInterpolatePoint, ST_LineSubstring

7.19.5 ST_LineSubstring

ST_LineSubstring — Returns the part of a line between two fractional locations.

Synopsis

geometry **ST_LineSubstring**(geometry a_linestring, float8 startfraction, float8 endfraction);
geography **ST_LineSubstring**(geography a_linestring, float8 startfraction, float8 endfraction);

Descrição

Computes the line which is the section of the input line starting and ending at the given fractional locations. The first argument must be a LINESTRING. The second and third arguments are values in the range [0, 1] representing the start and end locations as fractions of line length. The Z and M values are interpolated for added endpoints if present.

Se "início" e "fim" tiverem o mesmo valor, isso é equivalente a ST_LineInterpolatePoint.

Note
This only works with LINESTRINGs. To use on contiguous MULTILINESTRINGs first join them with ST_LineMerge.

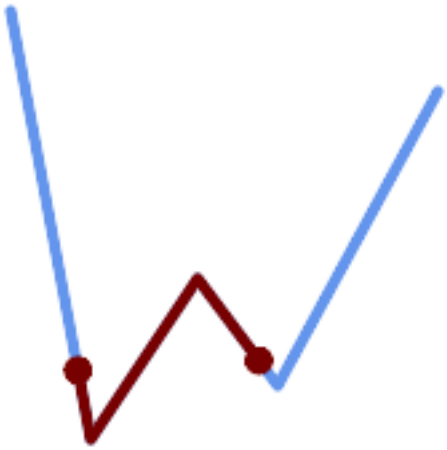
Note
Desde a liberação 1.1.1 essa função também interpola valores M e Z (quando presentes), enquanto as liberações anteriores configura eles para valores não específicos.

Disponibilidade: 1.1.0, Suporte a Z e M adicionado em 1.1.1

Alterações: 2.1.0 para 2.0.x foi chamada ST_Line_Substring.

 This function supports 3d and will not drop the z-index.

Exemplos



Uma linestring vista com 1/3 coberto (0.333, 0.666)

```
SELECT ST_AsText(ST_LineSubstring( 'LINESTRING (20 180, 50 20, 90 80, 120 40, 180 150)',  ↩
    0.333, 0.666));
------------------------------------------------------------------------------------------- ↩

LINESTRING (45.17311810399485 45.74337011202746, 50 20, 90 80, 112.97593050157862  ↩
    49.36542599789519)
```

If start and end locations are the same, the result is a POINT.

```
SELECT ST_AsText(ST_LineSubstring( 'LINESTRING(25 50, 100 125, 150 190)', 0.333, 0.333));
------------------------------------------
 POINT(69.2846934853974 94.2846934853974)
```

A query to cut a LineString into sections of length 100 or shorter. It uses `generate_series()` with a CROSS JOIN LATERAL to produce the equivalent of a FOR loop.

```
WITH data(id, geom) AS (VALUES
        ( 'A', 'LINESTRING( 0 0, 200 0)'::geometry ),
        ( 'B', 'LINESTRING( 0 100, 350 100)'::geometry ),
        ( 'C', 'LINESTRING( 0 200, 50 200)'::geometry )
    )
SELECT id, i,
       ST_AsText( ST_LineSubstring( geom, startfrac, LEAST( endfrac, 1 )) ) AS geom
FROM (
    SELECT id, geom, ST_Length(geom) len, 100 sublen FROM data
    ) AS d
CROSS JOIN LATERAL (
    SELECT i, (sublen * i) / len AS startfrac,
              (sublen * (i+1)) / len AS endfrac
    FROM generate_series(0, floor( len / sublen )::integer ) AS t(i)
    -- skip last i if line length is exact multiple of sublen
    WHERE (sublen * i) / len <> 1.0
    ) AS d2;

 id | i |            geom
----+---+----------------------------
 A  | 0 | LINESTRING(0 0,100 0)
 A  | 1 | LINESTRING(100 0,200 0)
 B  | 0 | LINESTRING(0 100,100 100)
 B  | 1 | LINESTRING(100 100,200 100)
 B  | 2 | LINESTRING(200 100,300 100)
 B  | 3 | LINESTRING(300 100,350 100)
 C  | 0 | LINESTRING(0 200,50 200)
```

Veja Também

ST_Length, ST_LineInterpolatePoint, ST_LineMerge

7.19.6 ST_LocateAlong

ST_LocateAlong — Returns the point(s) on a geometry that match a measure value.

Synopsis

geometria **ST_LocateAlong**(geometria ageom_with_measure, float8 a_measure, float8 offset);

Descrição

Returns the location(s) along a measured geometry that have the given measure values. The result is a Point or MultiPoint. Polygonal inputs are not supported.

If `offset` is provided, the result is offset to the left or right of the input line by the specified distance. A positive offset will be to the left, and a negative one to the right.

Note
Use this function only for linear geometries with an M component

The semantic is specified by the *ISO/IEC 13249-3 SQL/MM Spatial* standard.

Disponibilidade: 1.1.0 pelo nome antigo ST_Locate_Along_Measure.

Alterações: 2.0.0 nas versões anteriores era chamado de ST_Locate_Along_Measure. O nome antigo foi menosprezado e será removido no futuro, mas ainda está disponível.

 This function supports M coordinates.

 This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1.13

Exemplos

```
SELECT ST_AsText(
  ST_LocateAlong(
    'MULTILINESTRINGM((1 2 3, 3 4 2, 9 4 3),(1 2 3, 5 4 5))'::geometry,
    3 ));

----------------------------------
 MULTIPOINT M ((1 2 3),(9 4 3),(1 2 3))
```

Veja Também

ST_LocateBetween, ST_LocateBetweenElevations, ST_InterpolatePoint

7.19.7 ST_LocateBetween

ST_LocateBetween — Returns the portions of a geometry that match a measure range.

Synopsis

geometria **ST_LocateBetween**(geometria geomA, float8 measure_start, float8 measure_end, float8 offset);

Descrição

Retorna um valor de coleção de geometria derivado com elementos que combinam com a medida específica. Elementos polígonos não são suportados.

Se um deslocamento é fornecido, o resultado será o deslocamento para a direita ou para a esquerda da linha de entrada pelo número específico de unidades. Um deslocamento positivo será para a esquerda e um negativo para a direita.

Clipping a non-convex POLYGON may produce invalid geometry.

The semantic is specified by the *ISO/IEC 13249-3 SQL/MM Spatial* standard.

Disponibilidade: 1.1.0 pelo nome antigo ST_Locate_Between_Measures.

Alterações: 2.0.0 nas versões anteriores era chamado de ST_Locate_Along_Measure. O nome antigo foi menosprezado e será removido no futuro, mas ainda está disponível.

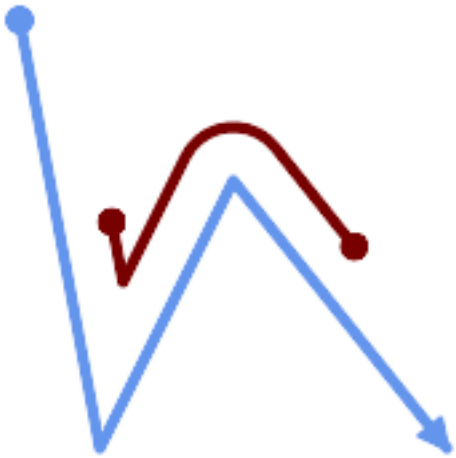
Enhanced: 3.0.0 - added support for POLYGON, TIN, TRIANGLE.

 This function supports M coordinates.

 This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1

Exemplos

```
SELECT ST_AsText(
  ST_LocateBetween(
       'MULTILINESTRING M ((1 2 3, 3 4 2, 9 4 3),(1 2 3, 5 4 5))':: geometry,
       1.5, 3 ));
------------------------------------------------------------------------
 GEOMETRYCOLLECTION M (LINESTRING M (1 2 3,3 4 2,9 4 3),POINT M (1 2 3))
```



A LineString with the section between measures 2 and 8, offset to the left

```
SELECT ST_AsText( ST_LocateBetween(
  ST_AddMeasure('LINESTRING (20 180, 50 20, 100 120, 180 20)', 0, 10),
  2, 8,
  20
));
------------------------------------------------------------------------
MULTILINESTRING((54.49835019899045 104.53426957938231,58.70056060327303  ↩
    82.12248075654186,69.16695286779743 103.05526528559065,82.11145618000168  ↩
    128.94427190999915,84.24893681714357 132.32493442618113,87.01636951231555  ↩
    135.21267035596549,90.30307285299679 137.49198684843182,93.97759758337769  ↩
    139.07172433557758,97.89298381958797 139.8887023914453,101.89263860095893  ↩
    139.9102465862721,105.81659870902816 139.13549527600819,109.50792827749828  ↩
    137.5954340631298,112.81899532549731 135.351656550512,115.6173761888606  ↩
    132.49390095108848,145.31017306064817 95.37790486135405))
```

Veja Também

ST_LocateAlong, ST_LocateAlong, ST_LocateBetween

7.19.8 ST_LocateBetweenElevations

ST_LocateBetweenElevations — Returns the portions of a geometry that lie in an elevation (Z) range.

Synopsis

geometria **ST_LocateBetweenElevations**(geometria geom_mline, float8 elevation_start, float8 elevation_end);

Descrição

Returns a geometry (collection) with the portions of a geometry that lie in an elevation (Z) range.

Clipping a non-convex POLYGON may produce invalid geometry.

Disponibilidade: 1.4.0

Enhanced: 3.0.0 - added support for POLYGON, TIN, TRIANGLE.

 This function supports 3d and will not drop the z-index.

Exemplos

```
SELECT ST_AsText(
  ST_LocateBetweenElevations(
    'LINESTRING(1 2 3, 4 5 6)'::geometry,
    2, 4 ));

             st_astext
-----------------------------------
 MULTILINESTRING Z ((1 2 3,2 3 4))

SELECT ST_AsText(
    ST_LocateBetweenElevations(
      'LINESTRING(1 2 6, 4 5 -1, 7 8 9)',
      6, 9)) As ewelev;

                                ewelev
----------------------------------------------------------------------
 GEOMETRYCOLLECTION Z (POINT Z (1 2 6),LINESTRING Z (6.1 7.1 6,7 8 9))
```

Veja Também

ST_Dump, ST_LocateAlong, ST_LocateBetween

7.19.9 ST_InterpolatePoint

ST_InterpolatePoint — Retorna o valor da dimensão de medida da geometria no ponto fechado para o ponto fornecido.

Synopsis

float8 **ST_InterpolatePoint**(geometry linear_geom_with_measure, geometry point);

Descrição

Returns an interpolated measure value of a linear measured geometry at the location closest to the given point.

Note
Use this function only for linear geometries with an M component

Disponibilidade: 2.0.0

 This function supports 3d and will not drop the z-index.

Exemplos

```
SELECT ST_InterpolatePoint('LINESTRING M (0 0 0, 10 0 20)', 'POINT(5 5)');
 --------------------
         10
```

Veja Também

ST_AddMeasure, ST_LocateAlong, ST_LocateBetween

7.19.10 ST_AddMeasure

ST_AddMeasure — Interpolates measures along a linear geometry.

Synopsis

geometria **ST_AddMeasure**(geometria geom_mline, float8 measure_start, float8 measure_end);

Descrição

Retorna uma geometria derivada com elementos de medida interpolados linearmente entre os pontos de início e de fim. Se a geometria não tem nenhuma dimensão de medida, uma é adicionada. Se a geometria tem dimensão de medida, é sobre escrita com novos valores. Somente LINESTRINGS e MULTILINESTRINGS são suportadas.

Disponibilidade: 1.5.0

 This function supports 3d and will not drop the z-index.

Exemplos

```
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))
```

7.20 Trajectory Functions

7.20.1 ST_IsValidTrajectory

ST_IsValidTrajectory — Tests if the geometry is a valid trajectory.

Synopsis

boolean **ST_IsValidTrajectory**(geometry line);

Description

Tests if a geometry encodes a valid trajectory. A valid trajectory is represented as a `LINESTRING` with measures (M values). The measure values must increase from each vertex to the next.

Valid trajectories are expected as input to spatio-temporal functions like ST_ClosestPointOfApproach

Availability: 2.2.0

 This function supports 3d and will not drop the z-index.

Examples

```
-- A valid trajectory
SELECT ST_IsValidTrajectory(ST_MakeLine(
  ST_MakePointM(0,0,1),
  ST_MakePointM(0,1,2))
);
 t

-- An invalid trajectory
SELECT ST_IsValidTrajectory(ST_MakeLine(ST_MakePointM(0,0,1), ST_MakePointM(0,1,0)));
NOTICE:  Measure of vertex 1 (0) not bigger than measure of vertex 0 (1)
 st_isvalidtrajectory
----------------------
 f
```

See Also

ST_ClosestPointOfApproach

7.20.2 ST_ClosestPointOfApproach

ST_ClosestPointOfApproach — Returns a measure at the closest point of approach of two trajectories.

Synopsis

float8 **ST_ClosestPointOfApproach**(geometry track1, geometry track2);

Description

Returns the smallest measure at which points interpolated along the given trajectories are at the smallest distance.

Inputs must be valid trajectories as checked by ST_IsValidTrajectory. Null is returned if the trajectories do not overlap in their M ranges.

See ST_LocateAlong for getting the actual points at the given measure.

Availability: 2.2.0

 This function supports 3d and will not drop the z-index.

Examples

```
-- Return the time in which two objects moving between 10:00 and 11:00
-- are closest to each other and their distance at that point
WITH inp AS ( SELECT
  ST_AddMeasure('LINESTRING Z (0 0 0, 10 0 5)'::geometry,
    extract(epoch from '2015-05-26 10:00'::timestamptz),
    extract(epoch from '2015-05-26 11:00'::timestamptz)
  ) a,
  ST_AddMeasure('LINESTRING Z (0 2 10, 12 1 2)'::geometry,
    extract(epoch from '2015-05-26 10:00'::timestamptz),
    extract(epoch from '2015-05-26 11:00'::timestamptz)
  ) b
), cpa AS (
  SELECT ST_ClosestPointOfApproach(a,b) m FROM inp
), points AS (
  SELECT ST_Force3DZ(ST_GeometryN(ST_LocateAlong(a,m),1)) pa,
         ST_Force3DZ(ST_GeometryN(ST_LocateAlong(b,m),1)) pb
  FROM inp, cpa
)
SELECT to_timestamp(m) t,
       ST_Distance(pa,pb) distance
FROM points, cpa;

               t               |     distance
-------------------------------+------------------
 2015-05-26 10:45:31.034483+02 | 1.96036833151395
```

See Also

ST_IsValidTrajectory, ST_DistanceCPA, ST_LocateAlong, ST_AddMeasure

7.20.3 ST_DistanceCPA

ST_DistanceCPA — Returns the distance between the closest point of approach of two trajectories.

Synopsis

float8 **ST_DistanceCPA**(geometry track1, geometry track2);

Description

Returns the minimum distance two moving objects have ever been each other.

Inputs must be valid trajectories as checked by ST_IsValidTrajectory. Null is returned if the trajectories do not overlap in their M ranges.

Availability: 2.2.0

 This function supports 3d and will not drop the z-index.

Examples

```
-- Return the minimum distance of two objects moving between 10:00 and 11:00
WITH inp AS ( SELECT
  ST_AddMeasure('LINESTRING Z (0 0 0, 10 0 5)'::geometry,
    extract(epoch from '2015-05-26 10:00'::timestamptz),
    extract(epoch from '2015-05-26 11:00'::timestamptz)
  ) a,
  ST_AddMeasure('LINESTRING Z (0 2 10, 12 1 2)'::geometry,
    extract(epoch from '2015-05-26 10:00'::timestamptz),
    extract(epoch from '2015-05-26 11:00'::timestamptz)
  ) b
)
SELECT ST_DistanceCPA(a,b) distance FROM inp;

     distance
------------------
 1.96036833151395
```

See Also

ST_IsValidTrajectory, ST_ClosestPointOfApproach, ST_AddMeasure, |=|

7.20.4 ST_CPAWithin

ST_CPAWithin — Tests if the closest point of approach of two trajectories is within the specified distance.

Synopsis

boolean **ST_CPAWithin**(geometry track1, geometry track2, float8 dist);

Description

Tests whether two moving objects have ever been closer than the specified distance.

Inputs must be valid trajectories as checked by ST_IsValidTrajectory. False is returned if the trajectories do not overlap in their M ranges.

Availability: 2.2.0

 This function supports 3d and will not drop the z-index.

Examples

```
WITH inp AS ( SELECT
  ST_AddMeasure('LINESTRING Z (0 0 0, 10 0 5)'::geometry,
    extract(epoch from '2015-05-26 10:00'::timestamptz),
    extract(epoch from '2015-05-26 11:00'::timestamptz)
  ) a,
  ST_AddMeasure('LINESTRING Z (0 2 10, 12 1 2)'::geometry,
    extract(epoch from '2015-05-26 10:00'::timestamptz),
    extract(epoch from '2015-05-26 11:00'::timestamptz)
  ) b
)
SELECT ST_CPAWithin(a,b,2), ST_DistanceCPA(a,b) distance FROM inp;

 st_cpawithin |     distance
--------------+------------------
 t            | 1.96521473776207
```

See Also

ST_IsValidTrajectory, ST_ClosestPointOfApproach, ST_DistanceCPA, |=|

7.21 SFCGAL Funções

7.21.1 postgis_sfcgal_version

postgis_sfcgal_version — retorna a versão do SFCGAL em uso

Synopsis

texto **postgis_sfcgal_version**(void);

Descrição

retorna a versão do SFCGAL em uso

Disponibilidade: 2.1.0

This method needs SFCGAL backend.

This function supports 3d and will not drop the z-index.

This function supports Polyhedral surfaces.

This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

Veja também

postgis_sfcgal_version

7.21.2 postgis_sfcgal_version

postgis_sfcgal_version — Returns the full version of SFCGAL in use including CGAL and Boost versions

Synopsis

texto **postgis_sfcgal_version**(void);

Descrição

Returns the full version of SFCGAL in use including CGAL and Boost versions

Disponibilidade: 2.1.0

 This method needs SFCGAL backend.

 This function supports 3d and will not drop the z-index.

 This function supports Polyhedral surfaces.

 This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

Veja também

postgis_sfcgal_version

7.21.3 ST_3DArea

ST_3DArea — Computa a área de geometrias de superfície 3D. Irá retornar 0 para sólidos.

Synopsis

float**ST_3DArea**(geometry geom1);

Descrição

Disponibilidade: 2.1.0

 This method needs SFCGAL backend.

 This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 8.1, 10.5

 This function supports 3d and will not drop the z-index.

 This function supports Polyhedral surfaces.

 This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

Exemplos

Nota: Por padrão uma superfície poliédrica construída de um WKT, é uma superfície de geometria, não sólida. Ela, portanto, tem uma área de superfície. Uma vez convertido em sólido, não tem nenhuma área.

```
SELECT ST_3DArea(geom) As cube_surface_area,
        ST_3DArea(ST_MakeSolid(geom)) As solid_surface_area
  FROM (SELECT '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)) )'::geometry) As f(geom);

 cube_surface_area | solid_surface_area
-------------------+--------------------
                 6 |                  0
```

Veja também

ST_Area, ST_MakeSolid, ST_IsSolid, ST_Area

7.21.4 ST_3DConvexHull

ST_3DConvexHull — Computa o eixo mediano aproximado de uma geometria territorial.

Synopsis

geometry **ST_3DConvexHull**(geometry geom1);

Descrição

Disponibilidade: 2.1.0

This method needs SFCGAL backend.

This function supports 3d and will not drop the z-index.

This function supports Polyhedral surfaces.

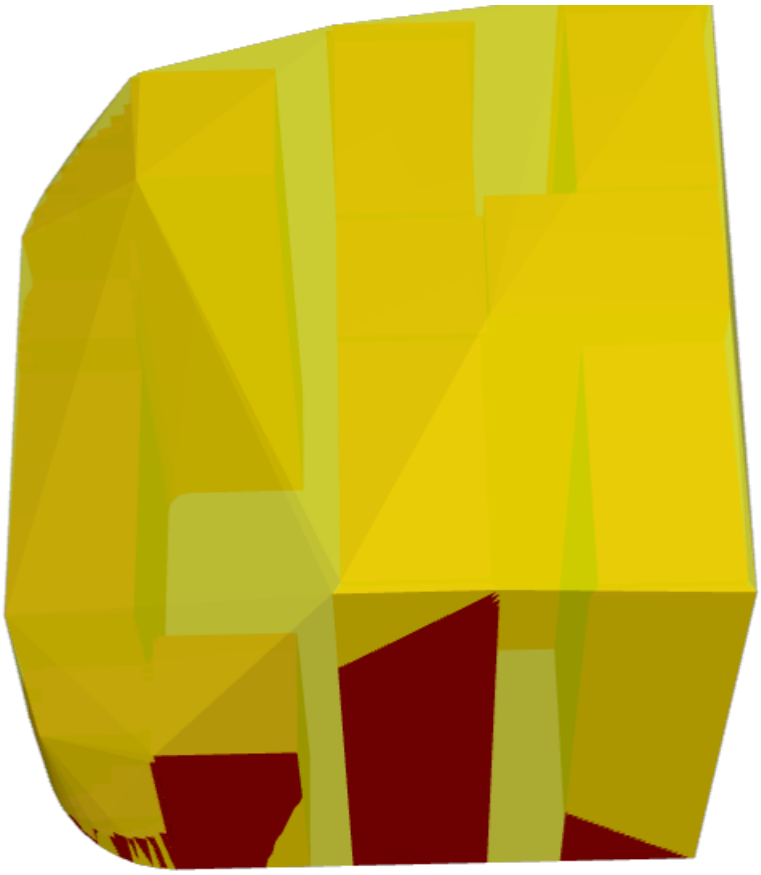
This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

Exemplos

```
SELECT ST_AsText(ST_3DConvexHull('LINESTRING Z(0 0 5, 1 5 3, 5 7 6, 9 5 3 , 5 7 5, 6 3 5) ↩
    '::geometry));
```

```
POLYHEDRALSURFACE Z (((1 5 3,9 5 3,0 0 5,1 5 3)),((1 5 3,0 0 5,5 7 6,1 5 3)),((5 7 6,5 7  ↩
    5,1 5 3,5 7 6)),((0 0 5,6 3 5,5 7 6,0 0 5)),((6 3 5,9 5 3,5 7 6,6 3 5)),((0 0 5,9 5 3,6  ↩
    3 5,0 0 5)),((9 5 3,5 7 5,5 7 6,9 5 3)),((1 5 3,5 7 5,9 5 3,1 5 3)))
```

```
WITH f AS (SELECT i, ST_Extrude(geom, 0,0, i ) AS geom
FROM ST_Subdivide(ST_Letters('CH'),5) WITH ORDINALITY AS sd(geom,i)
      )
      SELECT ST_3DConvexHull(ST_Collect(f.geom) )
      FROM f;
```



Original geometry overlaid with 3D convex hull

Veja também

ST_Letters, ST_AsX3D

7.21.5 ST_3DIntersection

ST_3DIntersection — Representar intersecção 3D

Synopsis

geometry **ST_3DIntersection**(geometry geom1, geometry geom2);

Descrição

Retorna uma geometria que é dividida entre geom1 e geom2

Disponibilidade: 2.1.0

This method needs SFCGAL backend.

This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1

This function supports 3d and will not drop the z-index.

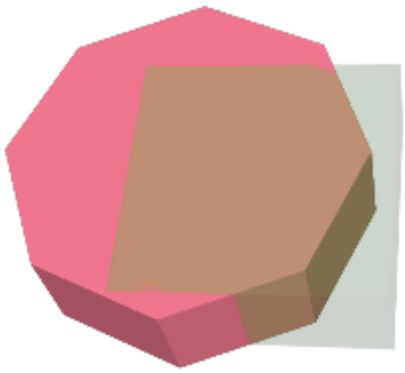
This function supports Polyhedral surfaces.

This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

Exemplos

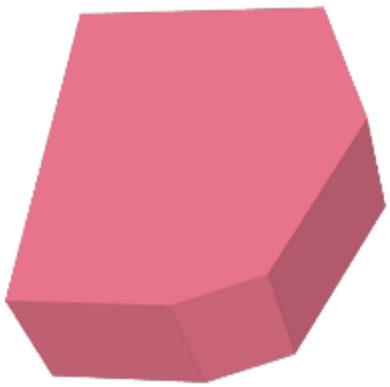
As imagens 3D foram criadas usando o PostGIS xref linkend="ST_AsX3D"/> e interpretadas no HTML usando: X3Dom HTML Javascript rendering library.

```
SELECT ST_Extrude(ST_Buffer( ↩
          ST_GeomFromText('POINT(100 90)'),
 50, 'quad_segs=2'),0,0,30) AS geom1,
        ST_Extrude(ST_Buffer( ↩
          ST_GeomFromText('POINT(80 80)'),
 50, 'quad_segs=1'),0,0,30) AS geom2;
```



Geometrias 3D originais cobertas. geom2 é apresentada semitransparente

```
SELECT ST_3DIntersection(geom1,geom2)
FROM ( SELECT ST_Extrude(ST_Buffer( ↩
          ST_GeomFromText('POINT(100 90)'),
 50, 'quad_segs=2'),0,0,30) AS geom1,
        ST_Extrude(ST_Buffer( ↩
          ST_GeomFromText('POINT(80 80)'),
 50, 'quad_segs=1'),0,0,30) AS geom2 ) As ↩
          t;
```



Intersecção de geom1 e geom2

Linestrings 3D e polígonos

```
SELECT ST_AsText(ST_3DIntersection(linestring, polygon)) As wkt
FROM  ST_GeomFromText('LINESTRING Z (2 2 6,1.5 1.5 7,1 1 8,0.5 0.5 8,0 0 10)') AS  ↩
    linestring
 CROSS JOIN ST_GeomFromText('POLYGON((0 0 8, 0 1 8, 1 1 8, 1 0 8, 0 0 8))') AS polygon;

              wkt
--------------------------------
 LINESTRING Z (1 1 8,0.5 0.5 8)
```

Cubo (superfície poliédrica fechada) e polígono Z

```
SELECT ST_AsText(ST_3DIntersection(
                ST_GeomFromText('POLYHEDRALSURFACE Z( ((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)) )'),
        'POLYGON Z ((0 0 0, 0 0 0.5, 0 0.5 0.5, 0 0.5 0, 0 0 0))'::geometry))
```

```
TIN Z (((0 0 0,0 0 0.5,0 0.5 0.5,0 0 0)),((0 0.5 0,0 0 0,0 0.5 0.5,0 0.5 0)))
```

Intersecção de 2 sólidos, que resulta em uma intersecção volumétrica, também é um sólido (ST_Dimension retorna 3)

```
SELECT ST_AsText(ST_3DIntersection( ST_Extrude(ST_Buffer('POINT(10 20)'::geometry,10,1) ↩
    ,0,0,30),
 ST_Extrude(ST_Buffer('POINT(10 20)'::geometry,10,1),2,0,10) ));
```

```
POLYHEDRALSURFACE Z (((13.3333333333333 13.3333333333333 10,20 20 0,20 20  ↩
    10,13.3333333333333 13.3333333333333 10)),
        ((20 20 10,16.6666666666667 23.3333333333333 10,13.3333333333333 13.3333333333333  ↩
            10,20 20 10)),
        ((20 20 0,16.6666666666667 23.3333333333333 10,20 20 10,20 20 0)),
        ((13.3333333333333 13.3333333333333 10,10 10 0,20 20 0,13.3333333333333  ↩
            13.3333333333333 10)),
        ((16.6666666666667 23.3333333333333 10,12 28 10,13.3333333333333 13.3333333333333  ↩
            10,16.6666666666667 23.3333333333333 10)),
        ((20 20 0,9.99999999999995 30 0,16.6666666666667 23.3333333333333 10,20 20 0)),
        ((10 10 0,9.99999999999995 30 0,20 20 0,10 10 0)),((13.3333333333333  ↩
            13.3333333333333 10,12 12 10,10 10 0,13.3333333333333 13.3333333333333 10)),
        ((12 28 10,12 12 10,13.3333333333333 13.3333333333333 10,12 28 10)),
        ((16.6666666666667 23.3333333333333 10,9.99999999999995 30 0,12 28  ↩
            10,16.6666666666667 23.3333333333333 10)),
        ((10 10 0,0 20 0,9.99999999999995 30 0,10 10 0)),
        ((12 12 10,11 11 10,10 10 0,12 12 10)),((12 28 10,11 11 10,12 12 10,12 28 10)),
        ((9.99999999999995 30 0,11 29 10,12 28 10,9.99999999999995 30 0)),((0 20 0,2 20  ↩
            10,9.99999999999995 30 0,0 20 0)),
        ((10 10 0,2 20 10,0 20 0,10 10 0)),((11 11 10,2 20 10,10 10 0,11 11 10)),((12 28  ↩
            10,11 29 10,11 11 10,12 28 10)),
        ((9.99999999999995 30 0,2 20 10,11 29 10,9.99999999999995 30 0)),((11 11 10,11 29  ↩
            10,2 20 10,11 11 10)))
```

7.21.6 ST_3DDifference

ST_3DDifference — Representar diferença 3D

Synopsis

geometry **ST_3DDifference**(geometry geom1, geometry geom2);

Descrição

Retorna aquela parte de geom1 que não faz parte de geom2.

Disponibilidade: 2.2.0

This method needs SFCGAL backend.

This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1

This function supports 3d and will not drop the z-index.

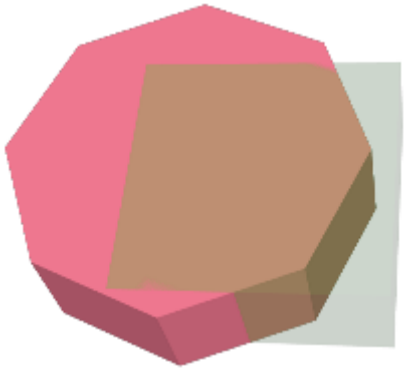
This function supports Polyhedral surfaces.

This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

Exemplos

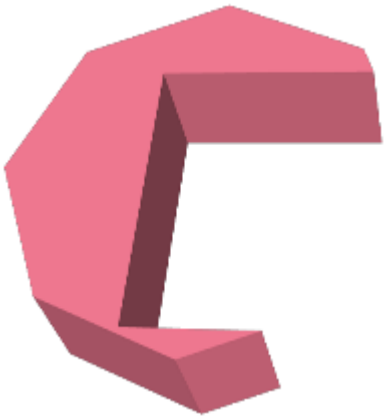
As imagens 3D foram criadas usando o PostGIS xref linkend="ST_AsX3D"/> e interpretadas no HTML usando: X3Dom HTML Javascript rendering library.

```
SELECT ST_Extrude(ST_Buffer( ↩
            ST_GeomFromText('POINT(100 90)'),
 50, 'quad_segs=2'),0,0,30) AS geom1,
        ST_Extrude(ST_Buffer( ↩
            ST_GeomFromText('POINT(80 80)'),
 50, 'quad_segs=1'),0,0,30) AS geom2;
```



Geometrias 3D originais cobertas. geom2 é a parte que será removida.

```
SELECT ST_3DDifference(geom1,geom2)
FROM ( SELECT ST_Extrude(ST_Buffer( ↩
            ST_GeomFromText('POINT(100 90)'),
 50, 'quad_segs=2'),0,0,30) AS geom1,
        ST_Extrude(ST_Buffer( ↩
            ST_GeomFromText('POINT(80 80)'),
 50, 'quad_segs=1'),0,0,30) AS geom2 ) As ↩
             t;
```



O que restou depois de remover geom2

Veja também

ST_Extrude, ST_AsX3D, ST_3DIntersection ST_3DUnion

7.21.7 ST_3DUnion

ST_3DUnion — Perform 3D union.

Synopsis

geometry **ST_3DUnion**(geometry geom1, geometry geom2);
geometry **ST_3DUnion**(geometry set g1field);

Descrição

Disponibilidade: 2.2.0

Availability: 3.3.0 aggregate variant was added

This method needs SFCGAL backend.

This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1

This function supports 3d and will not drop the z-index.

This function supports Polyhedral surfaces.

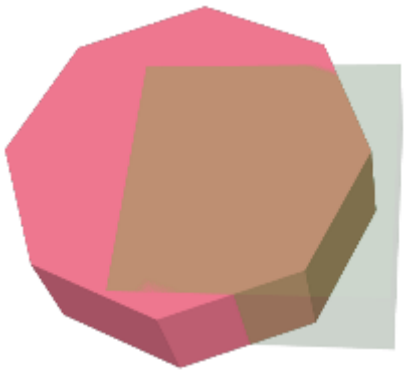
This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

Aggregate variant: returns a geometry that is the 3D union of a rowset of geometries. The ST_3DUnion() function is an "aggregate" function in the terminology of PostgreSQL. That means that it operates on rows of data, in the same way the SUM() and AVG() functions do and like most aggregates, it also ignores NULL geometries.

Exemplos

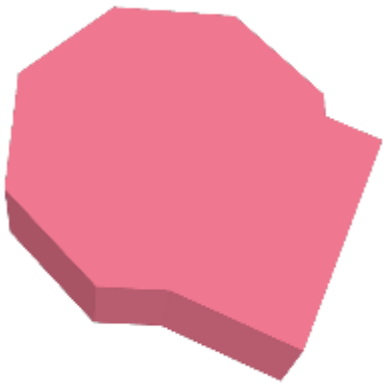
As imagens 3D foram criadas usando o PostGIS xref linkend="ST_AsX3D"/> e interpretadas no HTML usando: X3Dom HTML Javascript rendering library.

```
SELECT ST_Extrude(ST_Buffer( ↩
            ST_GeomFromText('POINT(100 90)'),
 50, 'quad_segs=2'),0,0,30) AS geom1,
       ST_Extrude(ST_Buffer( ↩
            ST_GeomFromText('POINT(80 80)'),
 50, 'quad_segs=1'),0,0,30) AS geom2;
```



Geometrias 3D originais cobertas. geom2 é a com transparência.

```
SELECT ST_3DUnion(geom1,geom2)
FROM ( SELECT ST_Extrude(ST_Buffer( ↩
            ST_GeomFromText('POINT(100 90)'),
 50, 'quad_segs=2'),0,0,30) AS geom1,
       ST_Extrude(ST_Buffer( ↩
            ST_GeomFromText('POINT(80 80)'),
 50, 'quad_segs=1'),0,0,30) AS geom2 ) As ↩
          t;
```



União de geom1 e geom2

Veja também

ST_Extrude, ST_AsX3D, ST_3DIntersection ST_3DDifference

7.21.8 ST_AlphaShape

ST_AlphaShape — Computes an Alpha-shape enclosing a geometry

Synopsis

geometry **ST_AlphaShape**(geometry geom, float alpha, boolean allow_holes = false);

Descrição

Computes the Alpha-Shape of the points in a geometry. An alpha-shape is a (usually) concave polygonal geometry which contains all the vertices of the input, and whose vertices are a subset of the input vertices. An alpha-shape provides a closer fit to the shape of the input than the shape produced by the convex hull.

The "closeness of fit" is controlled by the `alpha` parameter, which can have values from 0 to infinity. Smaller alpha values produce more concave results. Alpha values greater than some data-dependent value produce the convex hull of the input.

Note
Following the CGAL implementation, the alpha value is the *square* of the radius of the disc used in the Alpha-Shape algorithm to "erode" the Delaunay Triangulation of the input points. See CGAL Alpha-Shapes for more information. This is different from the original definition of alpha-shapes, which defines alpha as the radius of the eroding disc.

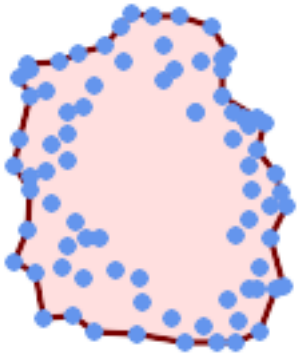
The computed shape does not contain holes unless the optional `allow_holes` argument is specified as true.

This function effectively computes a concave hull of a geometry in a similar way to ST_ConcaveHull, but uses CGAL and a different algorithm.

Availability: 3.3.0 - requires SFCGAL >= 1.4.1.

 This method needs SFCGAL backend.

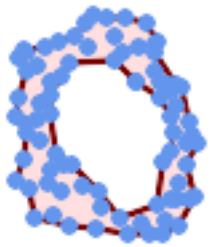
Exemplos



Alpha-shape of a MultiPoint (same example As ST_OptimalAlphaShape)

```
SELECT ST_AsText(ST_AlphaShape('MULTIPOINT((63 84),(76 88),(68 73),(53 18),(91 50),(81 70),
                (88 29),(24 82),(32 51),(37 23),(27 54),(84 19),(75 87),(44 42),(77 67),(90 ↩
                    30),(36 61),(32 65),
                (81 47),(88 58),(68 73),(49 95),(81 60),(87 50),
                (78 16),(79 21),(30 22),(78 43),(26 85),(48 34),(35 35),(36 40),(31 79),(83 ↩
                    29),(27 84),(52 98),(72 95),(85 71),
                (75 84),(75 77),(81 29),(77 73),(41 42),(83 72),(23 36),(89 53),(27 57),(57 ↩
                    97),(27 77),(39 88),(60 81),
                (80 72),(54 32),(55 26),(62 22),(70 20),(76 27),(84 35),(87 42),(82 54),(83 ↩
                    64),(69 86),(60 90),(50 86),(43 80),(36 73),
                (36 68),(40 75),(24 67),(23 60),(26 44),(28 33),(40 32),(43 19),(65 16),(73 ↩
                    16),(38 46),(31 59),(34 86),(45 90),(64 97))'::geometry,80.2));
```

```
POLYGON((89 53,91 50,87 42,90 30,88 29,84 19,78 16,73 16,65 16,53 18,43 19,37 23,30 22,28  ↩
    33,23 36,26 44,27 54,23 60,24 67,
          27 77,24 82,26 85,34 86,39 88,45 90,49 95,52 98,57 97,64 97,72 95,76 88,75 84,83  ↩
              72,85 71,88 58,89 53))
```



Alpha-shape of a MultiPoint, allowing holes (same example as ST_OptimalAlphaShape)

```
SELECT ST_AsText(ST_AlphaShape('MULTIPOINT((63 84),(76 88),(68 73),(53 18),(91 50),(81 70) ↩
    ,(88 29),(24 82),(32 51),(37 23),(27 54),(84 19),(75 87),(44 42),(77 67),(90 30),(36 61) ↩
    ,(32 65),(81 47),(88 58),(68 73),(49 95),(81 60),(87 50),
                (78 16),(79 21),(30 22),(78 43),(26 85),(48 34),(35 35),(36 40),(31 79),(83 ↩
                    29),(27 84),(52 98),(72 95),(85 71),
                (75 84),(75 77),(81 29),(77 73),(41 42),(83 72),(23 36),(89 53),(27 57),(57 ↩
                    97),(27 77),(39 88),(60 81),
                (80 72),(54 32),(55 26),(62 22),(70 20),(76 27),(84 35),(87 42),(82 54),(83 ↩
                    64),(69 86),(60 90),(50 86),(43 80),(36 73),
                (36 68),(40 75),(24 67),(23 60),(26 44),(28 33),(40 32),(43 19),(65 16),(73 ↩
                    16),(38 46),(31 59),(34 86),(45 90),(64 97))'::geometry, 100.1,true))
```

```
POLYGON((89 53,91 50,87 42,90 30,88 29,84 19,78 16,73 16,65 16,53 18,43 19,37 23,30 22,28  ↩
    33,23 36,26 44,27 54,23 60,24 67,27 77,24 82,26 85,34 86,39 88,45 90,49 95,52 98,57  ↩
    97,64 97,72 95,76 88,75 84,83 72,85 71,88 58,89 53),
              (36 61,36 68,40 75,43 80,50 86,60 81,68 73,77 67,81 60,82 54,81 47,78 43,76  ↩
                 27,62 22,54 32,48 34,44 42,38 46,36 61))
```

Alpha-shape of a MultiPoint, allowing holes (same example as ST_ConcaveHull)

```
SELECT ST_AlphaShape(
            'MULTIPOINT ((132 64), (114 64), (99 64), (81 64), (63 64), (57 49), (52 36),  ↩
                (46 20), (37 20), (26 20), (32 36), (39 55), (43 69), (50 84), (57 100), (63 ↩
                 118), (68 133), (74 149), (81 164), (88 180), (101 180), (112 180), (119  ↩
                164), (126 149), (132 131), (139 113), (143 100), (150 84), (157 69), (163  ↩
                51), (168 36), (174 20), (163 20), (150 20), (143 36), (139 49), (132 64),  ↩
                (99 151), (92 138), (88 124), (81 109), (74 93), (70 82), (83 82), (99 82),  ↩
                (112 82), (126 82), (121 96), (114 109), (110 122), (103 138), (99 151), (34 ↩
                 27), (43 31), (48 44), (46 58), (52 73), (63 73), (61 84), (72 71), (90 69) ↩
                , (101 76), (123 71), (141 62), (166 27), (150 33), (159 36), (146 44), (154 ↩
                 53), (152 62), (146 73), (134 76), (143 82), (141 91), (130 98), (126 104), ↩
                 (132 113), (128 127), (117 122), (112 133), (119 144), (108 147), (119 153) ↩
                , (110 171), (103 164), (92 171), (86 160), (88 142), (79 140), (72 124),  ↩
                (83 131), (79 118), (68 113), (63 102), (68 93), (35 45))'::geometry,102.2,  ↩
                true);
```

```
POLYGON((134 80,136 75,130 63,135 45,132 44,126 28,117 24,110 24,98 24,80 27,82 39,72 51,60 ↩
    48,56 34,52 52,42 50,
            34 54,39 66,40 81,34 90,36 100,40 116,36 123,39 128,51 129,58 132,68 135,74 ↩
                142,78 147,86 146,96 146,
            108 142,114 132,112 126,112 116,116 110,120 108,125 108,128 106,125 96,132  ↩
                87,134 80))
```

Veja também

ST_ConcaveHull, ST_OptimalAlphaShape

7.21.9 ST_ApproximateMedialAxis

ST_ApproximateMedialAxis — Computa o eixo mediano aproximado de uma geometria territorial.

Synopsis

geometria **ST_ApproximateMedialAxis**(geometria geom);

Descrição

Retorna um eixo mediano aproximado para a entrada territorial baseada do seu esqueleto reto. Usa uma API específica do SFCGAL quando construída contra uma versão capaz (1.2.0+). Senão a função é somente um wrapper em volta do ST_StraightSkeleton (caso menor).

Disponibilidade: 2.2.0

 This method needs SFCGAL backend.

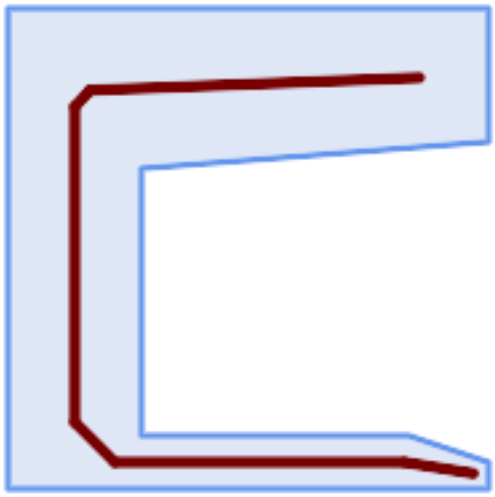
 This function supports 3d and will not drop the z-index.

 This function supports Polyhedral surfaces.

 This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

Exemplos

```
SELECT ST_ApproximateMedialAxis(ST_GeomFromText('POLYGON (( 190 190, 10 190, 10 10, 190 10, ↩
    190 20, 160 30, 60 30, 60 130, 190 140, 190 190 ))'));
```



Um polígono e seu eixo mediano aproximado

Veja também

ST_StraightSkeleton

7.21.10 ST_ConstrainedDelaunayTriangles

ST_ConstrainedDelaunayTriangles — Return a constrained Delaunay triangulation around the given input geometry.

Synopsis

geometry **ST_Tesselate**(geometry geom);

Descrição

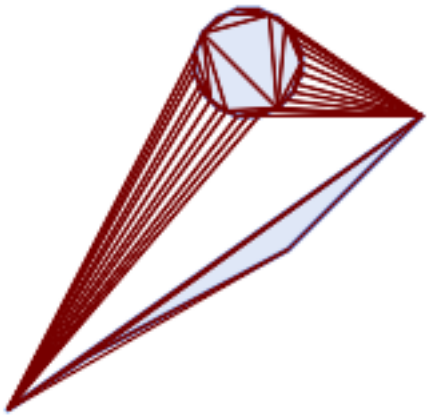
Return a Constrained Delaunay triangulation around the vertices of the input geometry. Output is a TIN.

This method needs SFCGAL backend.

Disponibilidade: 2.1.0

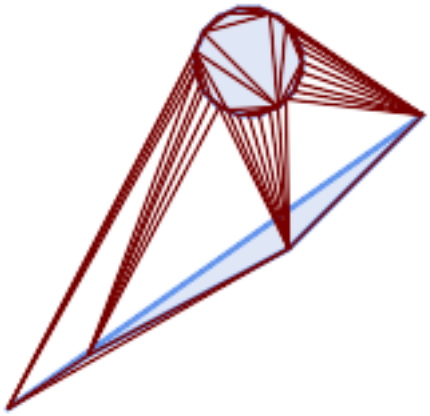
This function supports 3d and will not drop the z-index.

Exemplos



ST_ConstrainedDelaunayTriangles of 2 polygons

```
select ST_ConstrainedDelaunayTriangles(
                ST_Union(
                        'POLYGON((175 150, ↩
     20 40, 50 60, 125 100, 175 150))'::geometry,
                        ST_Buffer('POINT ↩
    (110 170)'::geometry, 20)
                    )
            );
```



ST_DelaunayTriangles of 2 polygons. Triangle edges cross polygon boundaries.

```
select ST_DelaunayTriangles(
                ST_Union(
                        'POLYGON((175 150, ↩
     20 40, 50 60, 125 100, 175 150))'::geometry,
                        ST_Buffer('POINT ↩
    (110 170)'::geometry, 20)
                    )
            );
```

Veja também

ST_DelaunayTriangles, ST_TriangulatePolygon, ST_Tesselate, ST_ConcaveHull, ST_Dump

7.21.11 ST_Extrude

ST_Extrude — Extrude uma superfície a um volume relacionado

Synopsis

geometry **ST_Extrude**(geometry geom, float x, float y, float z);

Descrição

Disponibilidade: 2.1.0

This method needs SFCGAL backend.

This function supports 3d and will not drop the z-index.

This function supports Polyhedral surfaces.

This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

Exemplos

As imagens 3D foram criadas usando o PostGIS xref linkend="ST_AsX3D"/> e interpretadas no HTML usando: X3Dom HTML Javascript rendering library.

```
SELECT ST_Buffer(ST_GeomFromText('POINT ↩
    (100 90)'),
  50, 'quad_segs=2'),0,0,30);
```

Octágono original formado a partir do ponto buffering

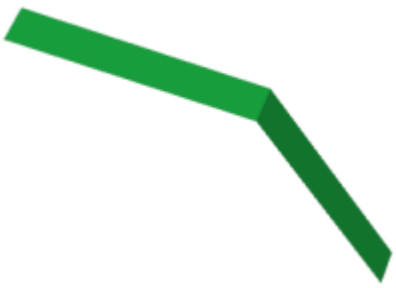
```
ST_Extrude(ST_Buffer(ST_GeomFromText(' ↩
    POINT(100 90)'),
 50, 'quad_segs=2'),0,0,30);
```

30 unidades expelidas do hexágono com Z produz uma PolyhedralSurfaceZ

```
SELECT ST_GeomFromText('LINESTRING(50 50, ↩
     100 90, 95 150)')
```

Linestring original

```
SELECT ST_Extrude(
 ST_GeomFromText('LINESTRING(50 50, 100  ↩
    90, 95 150)'),0,0,10));
```



Linestring expelida com Z produz uma PolyhedralSurfaceZ

Veja também

ST_AsX3D

7.21.12 ST_ForceLHR

ST_ForceLHR — Orientação força LHR

Synopsis

geometry **ST_ForceLHR**(geometry geom);

Descrição

Disponibilidade: 2.1.0

 This method needs SFCGAL backend.

 This function supports 3d and will not drop the z-index.

 This function supports Polyhedral surfaces.

 This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

7.21.13 ST_IsPlanar

ST_IsPlanar — Verifique se a superfície é ou não planar

Synopsis

boolean **ST_IsPlanar**(geometry geom);

Descrição

Disponibilidade: 2.2.0: Isso foi documentado em 2.1.0, mas foi deixado de fora acidentalmente na 2.1.

 This method needs SFCGAL backend.

 This function supports 3d and will not drop the z-index.

 This function supports Polyhedral surfaces.

 This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

7.21.14 ST_IsSolid

ST_IsSolid — teste se a geometria é um sólido. Nenhuma verificação de validade é representada.

Synopsis

boolean **ST_IsSolid**(geometry geom1);

Descrição

Disponibilidade: 2.2.0

 This method needs SFCGAL backend.

 This function supports 3d and will not drop the z-index.

 This function supports Polyhedral surfaces.

 This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

7.21.15 ST_MakeSolid

ST_MakeSolid — Molde a geometria para um sólido. Nenhuma verificação é apresentada. Para obter um sólido válido, a geometria de entrada deve ser uma superfície poliédrica fechada ou um TIN fechado.

Synopsis

geometry **ST_MakeSolid**(geometry geom1);

Descrição

Disponibilidade: 2.2.0

 This method needs SFCGAL backend.

 This function supports 3d and will not drop the z-index.

 This function supports Polyhedral surfaces.

 This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

7.21.16 ST_MinkowskiSum

ST_MinkowskiSum — Representar soma Minkowski

Synopsis

geometry **ST_MinkowskiSum**(geometry geom1, geometry geom2);

Descrição

Essa função representa uma soma minkowski 2D de um ponto, linha ou polígono com um polígono.

Uma soma minkowski de duas geometrias A e B é o conjunto de todos os pontos que estão somados a quaisquer pontos A e B. As somas minkowski são usadas em planos em movimento e design de ajuda de computadores. Maiores detalhes em Wikipedia Minkowski addition.

O primeiro parâmetro pode ser qualquer geometria 2D (ponto, linestring, polígono). Se uma geometria 3D é passada, ela será convertida para 2D forçando Z para 0, levando a possíveis casos de invalidade. O segundo parâmetro deve ser um polígono 2D.

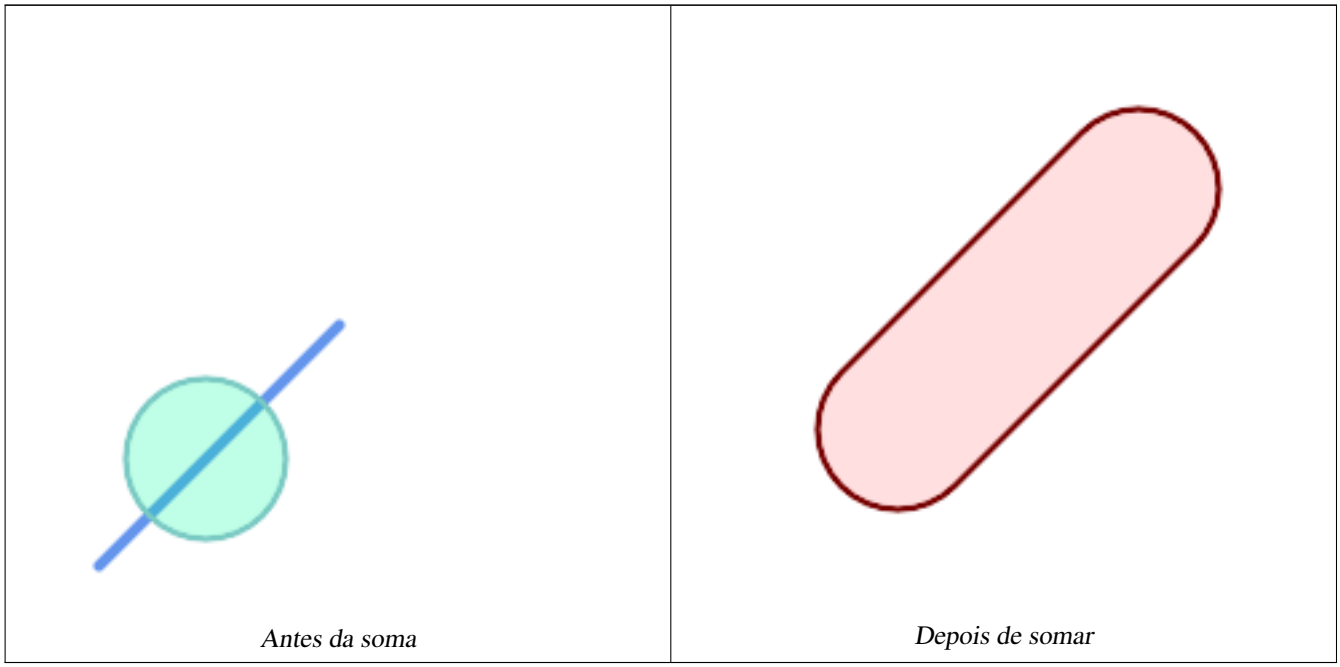
Implementação utiliza CGAL 2D Minkowskisum.

Disponibilidade: 2.1.0

 This method needs SFCGAL backend.

Exemplos

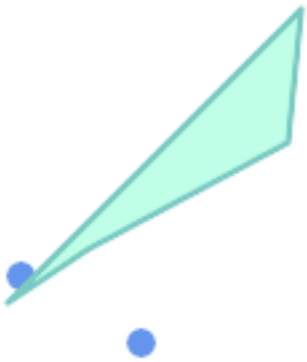
Soma minkowski de linestring e polígono circular onde a linestring corta através do círculo



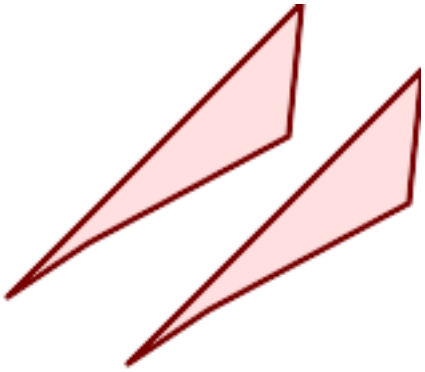
Antes da soma | *Depois de somar*

```
SELECT ST_MinkowskiSum(line, circle))
FROM (SELECT
    ST_MakeLine(ST_Point(10, 10),ST_Point(100, 100)) As line,
    ST_Buffer(ST_GeomFromText('POINT(50 50)'), 30) As circle) As foo;

-- wkt --
MULTIPOLYGON(((30 59.9999999999999,30.5764415879031 54.1472903395161,32.2836140246614  ↩
    48.5194970290472,35.0559116309237 43.3328930094119,38.7867965644036  ↩
    38.7867965644035,43.332893009412 35.0559116309236,48.5194970290474  ↩
    32.2836140246614,54.1472903395162 30.5764415879031,60.0000000000001 30,65.8527096604839  ↩
    30.5764415879031,71.4805029709527 32.2836140246614,76.6671069905881  ↩
    35.0559116309237,81.2132034355964 38.7867965644036,171.213203435596  ↩
    128.786796564404,174.944088369076 133.332893009412,177.716385975339  ↩
    138.519497029047,179.423558412097 144.147290339516,180 150,179.423558412097  ↩
    155.852709660484,177.716385975339 161.480502970953,174.944088369076  ↩
    166.667106990588,171.213203435596 171.213203435596,166.667106990588 174.944088369076,
161.480502970953 177.716385975339,155.852709660484 179.423558412097,150  ↩
    180,144.147290339516 179.423558412097,138.519497029047 177.716385975339,133.332893009412 ↩
     174.944088369076,128.786796564403 171.213203435596,38.7867965644035  ↩
    81.2132034355963,35.0559116309236 76.667106990588,32.2836140246614  ↩
    71.4805029709526,30.5764415879031 65.8527096604838,30 59.9999999999999)))
```

Soma minkowski de um polígono e multiponto



Antes da soma



Depois de somar: polígono é duplicado e transladado para posições de pontos

```
SELECT ST_MinkowskiSum(mp, poly)
FROM (SELECT 'MULTIPOINT(25 50,70 25)'::geometry As mp,
   'POLYGON((130 150, 20 40, 50 60, 125 100, 130 150))'::geometry As poly
    ) As foo


-- wkt --
MULTIPOLYGON(
    ((70 115,100 135,175 175,225 225,70 115)),
    ((120 65,150 85,225 125,275 175,120 65))
    )
```

7.21.17 ST_OptimalAlphaShape

ST_OptimalAlphaShape — Computes an Alpha-shape enclosing a geometry using an "optimal" alpha value.

Synopsis

geometry **ST_OptimalAlphaShape**(geometry geom, boolean allow_holes = false, integer nb_components = 1);

Descrição

Computes the "optimal" alpha-shape of the points in a geometry. The alpha-shape is computed using a value of α chosen so that:

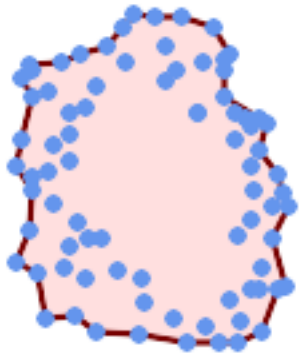
1. the number of polygon elements is equal to or smaller than `nb_components` (which defaults to 1)
2. all input points are contained in the shape

The result will not contain holes unless the optional `allow_holes` argument is specified as true.

Availability: 3.3.0 - requires SFCGAL >= 1.4.1.

This method needs SFCGAL backend.

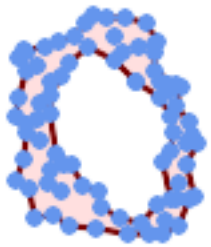
Exemplos



Optimal alpha-shape of a MultiPoint (same example as ST_AlphaShape)

```
SELECT ST_AsText(ST_OptimalAlphaShape('MULTIPOINT((63 84),(76 88),(68 73),(53 18),(91 50) ↩
    ,(81 70),
                (88 29),(24 82),(32 51),(37 23),(27 54),(84 19),(75 87),(44 42),(77 67),(90 ↩
                    30),(36 61),(32 65),
                (81 47),(88 58),(68 73),(49 95),(81 60),(87 50),
                (78 16),(79 21),(30 22),(78 43),(26 85),(48 34),(35 35),(36 40),(31 79),(83 ↩
                    29),(27 84),(52 98),(72 95),(85 71),
                (75 84),(75 77),(81 29),(77 73),(41 42),(83 72),(23 36),(89 53),(27 57),(57 ↩
                    97),(27 77),(39 88),(60 81),
                (80 72),(54 32),(55 26),(62 22),(70 20),(76 27),(84 35),(87 42),(82 54),(83 ↩
                    64),(69 86),(60 90),(50 86),(43 80),(36 73),
                (36 68),(40 75),(24 67),(23 60),(26 44),(28 33),(40 32),(43 19),(65 16),(73 ↩
                    16),(38 46),(31 59),(34 86),(45 90),(64 97))'::geometry));
```

```
POLYGON((89 53,91 50,87 42,90 30,88 29,84 19,78 16,73 16,65 16,53 18,43 19,37 23,30 22,28  ↩
    33,23 36,
                26 44,27 54,23 60,24 67,27 77,24 82,26 85,34 86,39 88,45 90,49 95,52 98,57  ↩
                    97,64 97,72 95,76 88,75 84,75 77,83 72,85 71,83 64,88 58,89 53))
```



Optimal alpha-shape of a MultiPoint, allowing holes (same example as ST_AlphaShape)

```
SELECT ST_AsText(ST_OptimalAlphaShape('MULTIPOINT((63 84),(76 88),(68 73),(53 18),(91 50) ↩
    ,(81 70),(88 29),(24 82),(32 51),(37 23),(27 54),(84 19),(75 87),(44 42),(77 67),(90 30) ↩
    ,(36 61),(32 65),(81 47),(88 58),(68 73),(49 95),(81 60),(87 50),
                (78 16),(79 21),(30 22),(78 43),(26 85),(48 34),(35 35),(36 40),(31 79),(83 ↩
                    29),(27 84),(52 98),(72 95),(85 71),
                (75 84),(75 77),(81 29),(77 73),(41 42),(83 72),(23 36),(89 53),(27 57),(57 ↩
                    97),(27 77),(39 88),(60 81),
                (80 72),(54 32),(55 26),(62 22),(70 20),(76 27),(84 35),(87 42),(82 54),(83 ↩
                    64),(69 86),(60 90),(50 86),(43 80),(36 73),
                (36 68),(40 75),(24 67),(23 60),(26 44),(28 33),(40 32),(43 19),(65 16),(73 ↩
                    16),(38 46),(31 59),(34 86),(45 90),(64 97))'::geometry, allow_holes =
> true));
```

```
POLYGON((89 53,91 50,87 42,90 30,88 29,84 19,78 16,73 16,65 16,53 18,43 19,37 23,30 22,28  ↩
    33,23 36,26 44,27 54,23 60,24 67,27 77,24 82,26 85,34 86,39 88,45 90,49 95,52 98,57  ↩
    97,64 97,72 95,76 88,75 84,75 77,83 72,85 71,83 64,88 58,89 53),(36 61,36 68,40 75,43  ↩
    80,50 86,60 81,68 73,77 67,81 60,82 54,81 47,78 43,81 29,76 27,70 20,62 22,55 26,54  ↩
    32,48 34,44 42,38 46,36 61))
```

Veja também

ST_ConcaveHull, ST_AlphaShape

7.21.18 ST_Orientation

ST_Orientation — Determine orientação da superfície

Synopsis

integer **ST_Orientation**(geometry geom);

Descrição

A função só se aplica a polígonos. Ela retorna -1 se o polígono estiver orientado no sentido anti-horário e 1 se estiver no sentido horário.

Disponibilidade: 2.1.0

This method needs SFCGAL backend.

This function supports 3d and will not drop the z-index.

7.21.19 ST_StraightSkeleton

ST_StraightSkeleton — Calcule um esqueleto em linha reta de uma geometria

Synopsis

geometry **ST_StraightSkeleton**(geometry geom);

Descrição

Disponibilidade: 2.1.0

This method needs SFCGAL backend.

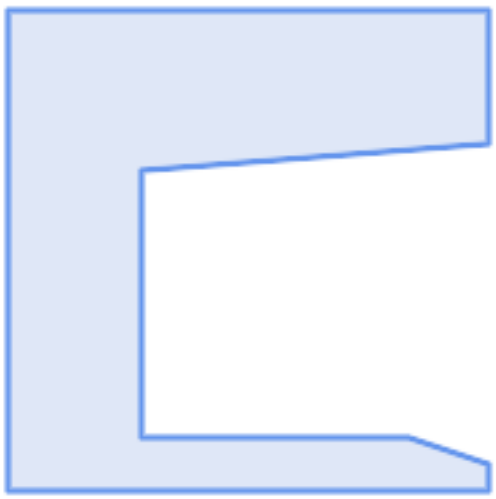
This function supports 3d and will not drop the z-index.

This function supports Polyhedral surfaces.

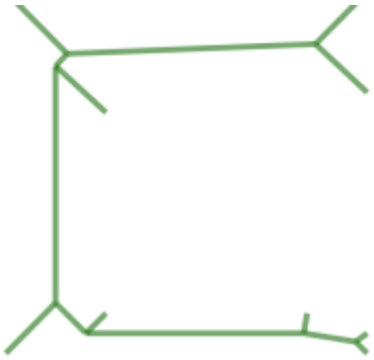
This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

Exemplos

```
SELECT ST_StraightSkeleton(ST_GeomFromText('POLYGON (( 190 190, 10 190, 10 10, 190 10, 190 ↩
    20, 160 30, 60 30, 60 130, 190 140, 190 190 ))'));
```



Polígono original



Esqueleto reto de polígono

7.21.20 ST_Tesselate

ST_Tesselate — Representa superfície tesselação de um polígono ou superfície poliédrica e retorna como uma TIN ou coleção de TINS

Synopsis

geometry **ST_Tesselate**(geometry geom);

Descrição

Usa como entrada uma superfície como um MULTI(POLÍGONO) ou SUPERFÍCIEPOLIÉDRICA e retorna uma representação TIN via o processo de tesselação usando triângulos.

Disponibilidade: 2.1.0

 This method needs SFCGAL backend.

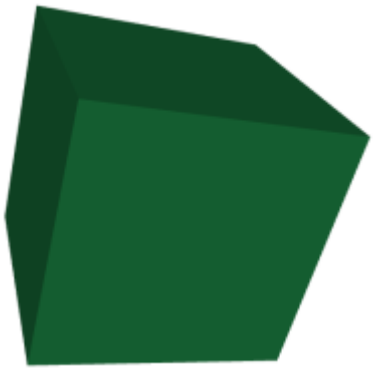
 This function supports 3d and will not drop the z-index.

 This function supports Polyhedral surfaces.

 This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

Exemplos

```
SELECT ST_GeomFromText('POLYHEDRALSURFACE Z( ((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)) )'));
```



Cubo original

```
SELECT ST_Tesselate(ST_GeomFromText('
    POLYHEDRALSURFACE Z( ((0 0 0, 0 0 1, 0 1 1, 0 1
        ((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_AsText output:

```
TIN Z (((0 0 0,0 0 1,0 1 1,0 0 0)),((0 1
    0,0 0 0,0 1 1,0 1 0)),
        ((0 0 0,0 1 0,1 1 0,0 0 0)),
        ((1 0 0,0 0 0,1 1 0,1 0 0)),((0 0
     1,1 0 0,1 0 1,0 0 1)),
        ((0 0 1,0 0 0,1 0 0,0 0 1)),
        ((1 1 0,1 1 1,1 0 1,1 1 0)),((1 0
     0,1 1 0,1 0 1,1 0 0)),
        ((0 1 0,0 1 1,1 1 1,0 1 0)),((1 1
     0,0 1 0,1 1 1,1 1 0)),
        ((0 1 1,1 0 1,1 1 1,0 1 1)),((0 1
     1,0 0 1,1 0 1,0 1 1)))
```

Cubo tesselado com triângulos coloridos

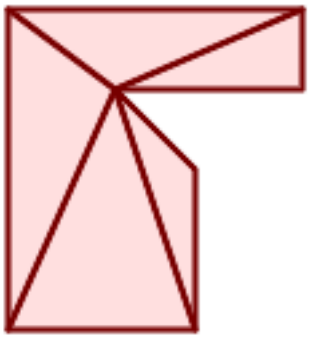
```
SELECT 'POLYGON (( 10 190, 10 70, 80 70,  ↩
    80 130, 50 160, 120 160, 120 190, 10 190 ))'::geometry;
```

Polígono original

```
SELECT
        ST_Tesselate('POLYGON (( 10 190,  ↩
    10 70, 80 70, 80 130, 50 160, 120 160, 120 190,
```

ST_AsText output

```
TIN(((80 130,50 160,80 70,80 130)),((50  ↩
    160,10 190,10 70,50 160)),
         ((80 70,50 160,10 70,80 70)) ↩
    ,((120 160,120 190,50 160,120 160)),
  ((120 190,10 190,50 160,120 190)))
```



Polígono tesselado

Veja também

ST_ConstrainedDelaunayTriangles, ST_DelaunayTriangles

7.21.21 ST_Volume

ST_Volume — Computa o volume de um sólido 3D. Se aplicado a geometrias com superfícies (mesmo fechadas), irão retornar 0.

Synopsis

float **ST_Volume**(geometry geom1);

Descrição

Disponibilidade: 2.2.0

This method needs SFCGAL backend.

This function supports 3d and will not drop the z-index.

This function supports Polyhedral surfaces.

This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 9.1 (same as ST_3DVolume)

Exemplo

Quando superfícies fechadas são criadas com WKT, elas são tratadas como territoriais ao invés de sólido. Para torná-las sólidos, você precis usar ST_MakeSolid. Geometrias territoriais não têm volume. Aqui está um exemplo demonstrativo.

```
SELECT ST_Volume(geom) As cube_surface_vol,
        ST_Volume(ST_MakeSolid(geom)) As solid_surface_vol
  FROM (SELECT '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)) )'::geometry) As f(geom);

 cube_surface_vol | solid_surface_vol
------------------+-------------------
                0 |                 1
```

Veja também

ST_3DArea, ST_MakeSolid, ST_IsSolid

7.22 Suporte de longas transações

Note
Os usuários devem usar serializable transaction level, senão o mecanismo de fechamento irá quebrar.

7.22.1 AddAuth

AddAuth — Adiciona um token de autorização para ser usado em transação atual.

Synopsis

boolean **AddAuth**(text auth_token);

Descrição

Adiciona um token de autorização para ser usado em transação atual.

Adds the current transaction identifier and authorization token to a temporary table called `temp_lock_have_table`.

Disponibilitade: 1.1.3

Exemplos

```
SELECT LockRow('towns', '353', 'priscilla');
                BEGIN TRANSACTION;
                        SELECT AddAuth('joey');
                        UPDATE towns SET geom = ST_Translate(geom,2,2) WHERE gid = 353;
                COMMIT;


                ---Error--
                ERROR:  UPDATE where "gid" = '353' requires authorization 'priscilla'
```

Veja também.

LockRow

7.22.2 CheckAuth

CheckAuth — Cria trigger em uma table para prevenir/permitir atualizações e exclusões de filas baseado em token de autorização.

Synopsis

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);

Descrição

Cria trigger em uma table para prevenir/permitir atualizações e exclusões de linhas baseado em token de autorização. identifica filas utilizando <rowid_col> column.

Se a_schema_name não passou, então, pesquise por tables no esquema atual.

Note
Se um trigger de autorização existe nessa table, função falha
Se o suporte de transação não estiver ativado, a função abre uma exceção.

Disponibilitade: 1.1.3

Exemplos

```
SELECT CheckAuth('public', 'towns', 'gid');
                        result
                        ------
                        0
```

Veja também.

AtivarLongasTransações

7.22.3 DesativarLongasTransações

DesativarLongasTransações — DesativarLongasTransações

Synopsis

text **DisableLongTransactions**();

Descrição

Desativa suporte de transações longas. Essa função remove as tables de metadados ad transação longa e derruba todos os triggers anexados às tables lock-checked.

Derruba a meta table chamada `authorization_table` e uma view chamada `authorized_tables` e todos os triggers chamados `checkauthtrigger`

Disponibilitade: 1.1.3

Exemplos

```
SELECT DisableLongTransactions();
--result--
Long transactions support disabled
```

Veja também.

AtivarLongasTransações

7.22.4 AtivarLongasTransações

AtivarLongasTransações — AtivarLongasTransações

Synopsis

text **EnableLongTransactions**();

Descrição

Ativa o suporte de longas transações. Essa função cria as tables de metadados que são requeridas, precisa ser chamada uma vez antes de usar outras funções nessa seção. Chamá-la duas vezes não tem problema.

Cria uma meta table chamada`authorization_table` e uma view chamada `authorized_tables`

Disponibilitade: 1.1.3

Exemplos

```
SELECT EnableLongTransactions();
--result--
Long transactions support enabled
```

Veja também.

DesativarLongasTransações

7.22.5 LockRow

LockRow — Configurar fechamento/autorização para uma fileira específica na table

Synopsis

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);

Descrição

Configurar fechamento/autorização para uma fileira específica na table <authid> é uma valor de texto, <expires> é uma marca temporal padrão para agora()+1hora. Retorna 1 se o fechamento tiver sido designado, 0 senão (já fechado por outra autorização)

Disponibilitade: 1.1.3

Exemplos

```
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
```

Veja também.

UnlockRows

7.22.6 UnlockRows

UnlockRows — Removes all locks held by an authorization token.

Synopsis

integer **UnlockRows**(text auth_token);

Descrição

Remove todos os fechamentos armazenados por id de autorização específica Retorna o número de fechamentos liberados.

Disponibilitade: 1.1.3

Exemplos

```
SELECT LockRow('towns', '353', 'priscilla');
                SELECT LockRow('towns', '2', 'priscilla');
                SELECT UnLockRows('priscilla');
                UnLockRows
                ------------
                2
```

Veja também.

LockRow

7.23 Version Functions

7.23.1 PostGIS_Extensions_Upgrade

PostGIS_Extensions_Upgrade — Packages and upgrades PostGIS extensions (e.g. postgis_raster, postgis_topology, postgis_sfcgal) to given or latest version.

Synopsis

text **PostGIS_Extensions_Upgrade**(text target_version=null);

Description

Packages and upgrades PostGIS extensions to given or latest version. Only extensions you have installed in the database will be packaged and upgraded if needed. Reports full PostGIS version and build configuration infos after. This is short-hand for doing multiple CREATE EXTENSION .. FROM unpackaged and ALTER EXTENSION .. UPDATE for each PostGIS extension. Currently only tries to upgrade extensions postgis, postgis_raster, postgis_sfcgal, postgis_topology, and postgis_tiger_geocoder.

Availability: 2.5.0

Note
Changed: 3.4.0 to add target_version argument.
Changed: 3.3.0 support for upgrades from any PostGIS version. Does not work on all systems.
Changed: 3.0.0 to repackage loose extensions and support postgis_raster.

Examples

```
SELECT PostGIS_Extensions_Upgrade();
```

```
NOTICE:  Packaging extension postgis
NOTICE:  Packaging extension postgis_raster
NOTICE:  Packaging extension postgis_sfcgal
NOTICE:  Extension postgis_topology is not available or not packagable for some reason
NOTICE:  Extension postgis_tiger_geocoder is not available or not packagable for some  ↩
    reason

                    postgis_extensions_upgrade
--------------------------------------------------------------------
 Upgrade completed, run SELECT postgis_full_version(); for details
(1 row)
```

See Also

Section 3.4, PostGIS_GEOS_Version, PostGIS_Lib_Version, PostGIS_LibXML_Version, PostGIS_PROJ_Version, PostGIS_Version

7.23.2 PostGIS_Full_Version

PostGIS_Full_Version — Reports full PostGIS version and build configuration infos.

Synopsis

text **PostGIS_Full_Version**();

Description

Reports full PostGIS version and build configuration infos. Also informs about synchronization between libraries and scripts suggesting upgrades as needed.

Enhanced: 3.4.0 now includes extra PROJ configurations NETWORK_ENABLED, URL_ENDPOINT and DATABASE_PATH of proj.db location

Examples

```
SELECT PostGIS_Full_Version();
                                                                    postgis_full_version
-------------------------------------------------------------------------------------------
POSTGIS="3.4.0dev 3.3.0rc2-993-g61bdf43a7" [EXTENSION] PGSQL="160" GEOS="3.12.0dev-CAPI ↩
    -1.18.0" SFCGAL="1.3.8" PROJ="7.2.1 NETWORK_ENABLED=OFF URL_ENDPOINT=https://cdn.proj. ↩
    org USER_WRITABLE_DIRECTORY=/tmp/proj DATABASE_PATH=/usr/share/proj/proj.db" GDAL="GDAL  ↩
    3.2.2, released 2021/03/05" LIBXML="2.9.10" LIBJSON="0.15" LIBPROTOBUF="1.3.3" WAGYU ↩
    ="0.5.0 (Internal)" TOPOLOGY RASTER
(1 row)
```

See Also

Section 3.4, PostGIS_GEOS_Version, PostGIS_Lib_Version, PostGIS_LibXML_Version, PostGIS_PROJ_Version, PostGIS_Wagyu_Ve
PostGIS_Version

7.23.3 PostGIS_GEOS_Version

PostGIS_GEOS_Version — Returns the version number of the GEOS library.

Synopsis

text **PostGIS_GEOS_Version**();

Description

Returns the version number of the GEOS library, or `NULL` if GEOS support is not enabled.

Examples

```
SELECT PostGIS_GEOS_Version();
 postgis_geos_version
----------------------
3.12.0dev-CAPI-1.18.0
(1 row)
```

See Also

PostGIS_Full_Version, PostGIS_Lib_Version, PostGIS_LibXML_Version, PostGIS_PROJ_Version, PostGIS_Version

7.23.4 PostGIS_GEOS_Compiled_Version

PostGIS_GEOS_Compiled_Version — Returns the version number of the GEOS library against which PostGIS was built.

Synopsis

text **PostGIS_GEOS_Compiled_Version**();

Description

Returns the version number of the GEOS library, or against which PostGIS was built.

Availability: 3.4.0

Examples

```
SELECT PostGIS_GEOS_Compiled_Version();
 postgis_geos_compiled_version
-------------------------------
 3.12.0
(1 row)
```

See Also

PostGIS_GEOS_Version, PostGIS_Full_Version

7.23.5 PostGIS_Liblwgeom_Version

PostGIS_Liblwgeom_Version — Returns the version number of the liblwgeom library. This should match the version of PostGIS.

Synopsis

text **PostGIS_Liblwgeom_Version**();

Description

Returns the version number of the liblwgeom library/

Examples

```
SELECT PostGIS_Liblwgeom_Version();
postgis_liblwgeom_version
-------------------------
3.4.0dev 3.3.0rc2-993-g61bdf43a7
(1 row)
```

See Also

PostGIS_Full_Version, PostGIS_Lib_Version, PostGIS_LibXML_Version, PostGIS_PROJ_Version, PostGIS_Version

7.23.6 PostGIS_LibXML_Version

PostGIS_LibXML_Version — Returns the version number of the libxml2 library.

Synopsis

text **PostGIS_LibXML_Version**();

Description

Returns the version number of the LibXML2 library.

Availability: 1.5

Examples

```
SELECT PostGIS_LibXML_Version();
 postgis_libxml_version
----------------------
 2.9.10
(1 row)
```

See Also

PostGIS_Full_Version, PostGIS_Lib_Version, PostGIS_PROJ_Version, PostGIS_GEOS_Version, PostGIS_Version

7.23.7 PostGIS_Lib_Build_Date

PostGIS_Lib_Build_Date — Returns build date of the PostGIS library.

Synopsis

text **PostGIS_Lib_Build_Date**();

Description

Returns build date of the PostGIS library.

Examples

```
SELECT PostGIS_Lib_Build_Date();
 postgis_lib_build_date
------------------------
 2023-06-22 03:56:11
(1 row)
```

7.23.8 PostGIS_Lib_Version

PostGIS_Lib_Version — Returns the version number of the PostGIS library.

Synopsis

text **PostGIS_Lib_Version**();

Description

Returns the version number of the PostGIS library.

Examples

```
SELECT PostGIS_Lib_Version();
 postgis_lib_version
---------------------
 3.4.0dev
(1 row)
```

See Also

PostGIS_Full_Version, PostGIS_GEOS_Version, PostGIS_LibXML_Version, PostGIS_PROJ_Version, PostGIS_Version

7.23.9 PostGIS_PROJ_Version

PostGIS_PROJ_Version — Returns the version number of the PROJ4 library.

Synopsis

text **PostGIS_PROJ_Version**();

Description

Returns the version number of the PROJ library and some configuration options of proj.

Enhanced: 3.4.0 now includes NETWORK_ENABLED, URL_ENDPOINT and DATABASE_PATH of proj.db location

Examples

```
SELECT PostGIS_PROJ_Version();
  postgis_proj_version
-------------------------
7.2.1 NETWORK_ENABLED=OFF URL_ENDPOINT=https://cdn.proj.org USER_WRITABLE_DIRECTORY=/tmp/ ↩
    proj DATABASE_PATH=/usr/share/proj/proj.db
(1 row)
```

See Also

PostGIS_Full_Version, PostGIS_GEOS_Version, PostGIS_Lib_Version, PostGIS_LibXML_Version, PostGIS_Version

7.23.10 PostGIS_Wagyu_Version

PostGIS_Wagyu_Version — Returns the version number of the internal Wagyu library.

Synopsis

text **PostGIS_Wagyu_Version**();

Description

Returns the version number of the internal Wagyu library, or `NULL` if Wagyu support is not enabled.

Examples

```
SELECT PostGIS_Wagyu_Version();
 postgis_wagyu_version
-----------------------
 0.5.0 (Internal)
(1 row)
```

See Also

PostGIS_Full_Version, PostGIS_GEOS_Version, PostGIS_PROJ_Version, PostGIS_Lib_Version, PostGIS_LibXML_Version, PostGIS_Version

7.23.11 PostGIS_Scripts_Build_Date

PostGIS_Scripts_Build_Date — Returns build date of the PostGIS scripts.

Synopsis

text **PostGIS_Scripts_Build_Date**();

Description

Returns build date of the PostGIS scripts.

Availability: 1.0.0RC1

Examples

```
SELECT PostGIS_Scripts_Build_Date();
  postgis_scripts_build_date
-------------------------
 2023-06-22 03:56:11
(1 row)
```

See Also

PostGIS_Full_Version, PostGIS_GEOS_Version, PostGIS_Lib_Version, PostGIS_LibXML_Version, PostGIS_Version

7.23.12 PostGIS_Scripts_Installed

PostGIS_Scripts_Installed — Returns version of the PostGIS scripts installed in this database.

Synopsis

text **PostGIS_Scripts_Installed**();

Description

Returns version of the PostGIS scripts installed in this database.

Note
If the output of this function doesn't match the output of PostGIS_Scripts_Released you probably missed to properly upgrade an existing database. See the Upgrading section for more info.

Availability: 0.9.0

Examples

```
SELECT PostGIS_Scripts_Installed();
  postgis_scripts_installed
-------------------------
 3.4.0dev 3.3.0rc2-993-g61bdf43a7
(1 row)
```

See Also

PostGIS_Full_Version, PostGIS_Scripts_Released, PostGIS_Version

7.23.13 PostGIS_Scripts_Released

PostGIS_Scripts_Released — Returns the version number of the postgis.sql script released with the installed PostGIS lib.

Synopsis

text **PostGIS_Scripts_Released**();

Description

Returns the version number of the postgis.sql script released with the installed PostGIS lib.

Note
Starting with version 1.1.0 this function returns the same value of PostGIS_Lib_Version. Kept for backward compatibility.

Availability: 0.9.0

Examples

```
SELECT PostGIS_Scripts_Released();
  postgis_scripts_released
--------------------------
 3.4.0dev 3.3.0rc2-993-g61bdf43a7
(1 row)
```

See Also

PostGIS_Full_Version, PostGIS_Scripts_Installed, PostGIS_Lib_Version

7.23.14 PostGIS_Version

PostGIS_Version — Returns PostGIS version number and compile-time options.

Synopsis

text **PostGIS_Version**();

Description

Returns PostGIS version number and compile-time options.

Examples

```
SELECT PostGIS_Version();
                        postgis_version
---------------------------------------
 3.4 USE_GEOS=1 USE_PROJ=1 USE_STATS=1
(1 row)
```

See Also

PostGIS_Full_Version, PostGIS_GEOS_Version, PostGIS_Lib_Version, PostGIS_LibXML_Version, PostGIS_PROJ_Version

7.24 Grandes Variáveis Unificadas Personalizadas do PostGIS (GUCs)

7.24.1 postgis.backend

postgis.backend — O backend para fazer a manutenção de uma função onde GEOS e SFCGAL sobrepõe. Opções: geos ou sfcgal. Padrão para geos.

Descrição

Essa GUC só é relevante se você compilou o PostGIS com o suporte sfcgal. Por padrão o backend `geos` é usado por funções onde o GEOS e o SFCGAL têm o mesmo nome. Essa variável permite exceder e fazer o sfcgal ser o backend para a solicitação do serviço.

Disponibilidade: 2.1.0

Exemplos

Configura backend apenas para vida de conexão

```
set postgis.backend = sfcgal;
```

Configura backend para novas conexões para o banco de dados

```
ALTER DATABASE mygisdb SET postgis.backend = sfcgal;
```

Veja também.

Section 7.21

7.24.2 postgis.gdal_datapath

postgis.gdal_datapath — Uma opção de configuração para designar o valor da opção GDAL_DATA do GDAL. Se não funcionar, a variável ambiental GDAL_DATA é usada.

Descrição

Uma variável GUC do PostgreSQL para configurar o valor da opção GDAL_DATA do GDAL. O valor `postgis.gdal_datapath` deve ser o path físico completo para os arquivos de dados do GDAL.

Essa opção de configuração é mais usada para plataformas do Windows, onde os arquivos de dados path do GDAL's não estão hard-coded. Essa opção deve também ser configurada quando esses aquivos não estiverem no path esperado.

Note
Essa opção pode ser configurada no arquivo de configuração postgresql.conf. Pode ser configurado por conexão ou transação.

Disponibilidade: 2.2.0

Note
Informação adicional sobre o GDAL_DATA está disponível em GDAL's Configuration Options.

Exemplos

Configurar e resetar `postgis.gdal_datapath`

```
SET postgis.gdal_datapath TO '/usr/local/share/gdal.hidden';
SET postgis.gdal_datapath TO default;
```

Configurando no Windows para um banco de dados específico

```
ALTER DATABASE gisdb
SET postgis.gdal_datapath = 'C:/Program Files/PostgreSQL/9.3/gdal-data';
```

Veja também.

PostGIS_GDAL_Version, ST_Transform

7.24.3 postgis.gdal_enabled_drivers

postgis.gdal_enabled_drivers — Uma opção de configuração para estabelecer os drivers GDAL ativados no ambiente POstGIS. Afeta a variável GDAL_SKIP do GDAL.

Descrição

Uma opção de configuração para estabelecer os drivers GDAL ativados no PostGIS. Afeta a variável de configuração GDAL_SKIP. Essa opção pode ser estabelecida no arquivo de configuração do PostgreSQL: postgresql.conf. Ela também pode ser estabelecida por conexão ou transação.

O valor inicial do `postgis.gdal_enabled_drivers` também pode ser estabelecido passando a variável de ambiente `POSTGIS_GDAL_ENABLED_DRIVERS` com a lista de drivers ativados para o processo de começar o PostgreSQL.

Dispositivos ativados específicos GDAL podem ser especificados pelos dispositivos de nome ou código curto. Dispositivos com nomes ou códigos curtos podem ser encontrados emGDAL Raster Formats. Vários dispositivos podem ser encontrados, colocando um espaço entre cada um deles.

Note

Existem três códigos especiais disponíveis para `postgis.gdal_enabled_drivers`. Os códigos são case-sensitive.

- `DISABLE_ALL` desabilita todos os drivers GDAL. Se presente, `DISABLE_ALL` excede todos os outros valores em `postgis.gdal_enabled_drivers`.
- `ENABLE_ALL` ativa todos os drivers GDAL.
- `VSICURL` ativa o arquivo do sistema virtual `/vsicurl/` do GDAL.

Quando `postgis.gdal_enabled_drivers` é configurado para DESABILITAR_TODOS, tenta usar out-db rasters, ST_FromGDALRaster(), ST_AsGDALRaster(), ST_AsTIFF(), ST_AsJPEG() e ST_AsPNG() resultará em mensagens de erro.

Note

Na instalação padrão do PostGIS, `postgis.gdal_enabled_drivers` é configurado para DESABILITAR_TODOS.

Note
Informações adicionais sobre GDAL_SKIP estão disponíveis em Opções de Configuração.

Disponibilidade: 2.2.0

Exemplos

Configurar e resetar `postgis.gdal_enabled_drivers`

Configura backend para todas as novas conexões para o banco de dados

```
ALTER DATABASE mygisdb SET postgis.gdal_enabled_drivers TO 'GTiff PNG JPEG';
```

Estabelece drivers ativados padrões para todas as conexões para fazer a manutenção. Requer acesso super do usuário e PostgreSQL 9.4+. Aquele banco de dados, sessão e usuário não excedem isso.

```
ALTER SYSTEM SET postgis.gdal_enabled_drivers TO 'GTiff PNG JPEG';
SELECT pg_reload_conf();
```

```
SET postgis.gdal_enabled_drivers TO 'GTiff PNG JPEG';
SET postgis.gdal_enabled_drivers = default;
```

Ativar todos os dispositivos GDAL

```
SET postgis.gdal_enabled_drivers = 'ENABLE_ALL';
```

Desativar todos os dispositivos GDAL

```
SET postgis.gdal_enabled_drivers = 'DISABLE_ALL';
```

Veja também.

ST_FromGDALRaster, ST_AsGDALRaster, ST_AsTIFF, ST_AsPNG, ST_AsJPEG, postgis.enable_outdb_rasters

7.24.4 postgis.enable_outdb_rasters

postgis.enable_outdb_rasters — Uma opção de configuração booleana para ativar o acesso ao out-db raster bands.

Descrição

Uma opção de configuração booleana para ativar o acesso ao ut-db raster bands. Essa opção pode ser estabelecida no arquivo de configuração: postgresql.conf. Ela também pode ser estabelecida por conexão ou transação.

O valor inicial de `postgis.enable_outdb_rasters` também pode ser estabelecido passando a variável de ambiente `POSTGIS_ENABLE_OUTDB_RASTERS` com um valor não-zero para o processo de começar o PostgreSQL.

Note
Mesmo se `postgis.enable_outdb_rasters` is é verdade, o GUC `postgis.enable_outdb_rasters` determina os formatos raster acessíveis.

Note
Na instalação padrão do PostGIS, `postgis.enable_outdb_rasters` é colocado como Falso.

Disponibilidade: 2.2.0

Exemplos

Configurar e resetar `postgis.enable_outdb_rasters`

```
SET postgis.enable_outdb_rasters TO True;
SET postgis.enable_outdb_rasters = default;
SET postgis.enable_outdb_rasters = True;
SET postgis.enable_outdb_rasters = False;
```

Set for specific database

```
ALTER DATABASE gisdb SET postgis.enable_outdb_rasters = true;
```

Setting for whole database cluster. You need to reconnect to the database for changes to take effect.

```
--writes to postgres.auto.conf
ALTER SYSTEM postgis.enable_outdb_rasters = true;
 --Reloads postgres conf
SELECT pg_reload_conf();
```

Veja também.

postgis.gdal_enabled_drivers postgis.gdal_datapath

7.24.5 postgis.gdal_datapath

postgis.gdal_datapath — Uma opção de configuração booleana para ativar o acesso ao out-db raster bands.

Descrição

A string configuration to set options used when working with an out-db raster. Configuration options control things like how much space GDAL allocates to local data cache, whether to read overviews, and what access keys to use for remote out-db data sources.

Disponibilidade: 2.2.0

Exemplos

Configurar e resetar `postgis.enable_outdb_rasters`

```
SET postgis.gdal_config_options = 'AWS_ACCESS_KEY_ID=xxxxxxxxxxxxxxx AWS_SECRET_ACCESS_KEY= ↩
    yyyyyyyyyyyyyyyyyyyyyyyyyy';
```

Set `postgis.gdal_vsi_options` just for the *current transaction* using the `LOCAL` keyword:

```
SET LOCAL postgis.gdal_config_options = 'AWS_ACCESS_KEY_ID=xxxxxxxxxxxxxxx  ↩
    AWS_SECRET_ACCESS_KEY=yyyyyyyyyyyyyyyyyyyyyyyyyy';
```

Veja também.

postgis.enable_outdb_rasters postgis.gdal_enabled_drivers

7.25 Troubleshooting Functions

7.25.1 PostGIS_AddBBox

PostGIS_AddBBox — Add bounding box to the geometry.

Synopsis

geometry **PostGIS_AddBBox**(geometry geomA);

Description

Add bounding box to the geometry. This would make bounding box based queries faster, but will increase the size of the geometry.

Note
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.

 This method supports Circular Strings and Curves.

Examples

```
UPDATE sometable
 SET geom =  PostGIS_AddBBox(geom)
 WHERE PostGIS_HasBBox(geom) = false;
```

See Also

PostGIS_DropBBox, PostGIS_HasBBox

7.25.2 PostGIS_DropBBox

PostGIS_DropBBox — Drop the bounding box cache from the geometry.

Synopsis

geometry **PostGIS_DropBBox**(geometry geomA);

Description

Drop the bounding box cache from the geometry. This reduces geometry size, but makes bounding-box based queries slower. It is also used to drop a corrupt bounding box. A tale-tell sign of a corrupt cached bounding box is when your ST_Intersects and other relation queries leave out geometries that rightfully should return true.

Note
Bounding boxes are automatically added to geometries and improve speed of queries 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. This kind of corruption has been observed in 8.3-8.3.6 series whereby cached bboxes were not always recalculated when a geometry changed and upgrading to a newer version without a dump reload will not correct already corrupted boxes. So one can manually correct using below and readd the bbox or do a dump reload.

 This method supports Circular Strings and Curves.

Examples

```
--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 geom =  PostGIS_DropBBox(geom)
 WHERE Not (Box2D(ST_AsBinary(geom)) = Box2D(geom));

        UPDATE sometable
 SET geom =  PostGIS_AddBBox(geom)
 WHERE Not PostGIS_HasBBOX(geom);
```

See Also

PostGIS_AddBBox, PostGIS_HasBBox, Box2D

7.25.3 PostGIS_HasBBox

PostGIS_HasBBox — Returns TRUE if the bbox of this geometry is cached, FALSE otherwise.

Synopsis

boolean **PostGIS_HasBBox**(geometry geomA);

Description

Returns TRUE if the bbox of this geometry is cached, FALSE otherwise. Use PostGIS_AddBBox and PostGIS_DropBBox to control caching.

 This method supports Circular Strings and Curves.

Examples

```
SELECT geom
FROM sometable WHERE PostGIS_HasBBox(geom) = false;
```

See Also

PostGIS_AddBBox, PostGIS_DropBBox

Chapter 8

Topologia

Os tipos e as funções de topologia do PostGIS são usados para administrar objetos como: faces, bordas e nodos.

Sandro Santilli's presentation at PostGIS Day Paris 2011 conference gives a good synopsis of PostGIS Topology and where it is headed Topology with PostGIS 2.0 slide deck.

Vincent Picavet provides a good synopsis and overview of what is Topology, how is it used, and various FOSS4G tools that support it in PostGIS Topology PGConf EU 2012.

Um exemplo de um banco de dados GIS baseado topologicamente é o banco de dados US Census Topologically Integrated Geographic Encoding and Referencing System (TIGER). Se você quiser experimentar com a topologia POstGIS e precisa de alguns dados, confira Topology_Load_Tiger.

O módulo PostGIS Topologia existiu em versões anteriores, mas nunca foi parte da documentação Oficial do PostGIS. A maior limpeza PostGIS 2.0.0, vai remover todas as funções menores, consertar problemas de usabilidade, vai documentar melhor as características e funções e melhorar a conformidade com os padrões SQL-MM.

Detalhes deste projeto podem ser encontrados em PostGIS Topology Wiki

Todas as funções e tables associadas com este módulo estão instaladas em um esquema nomeado `topology`.

Funções que são definidas no padrão SQL/MM estão prefixadas com ST_ e funções específicas para o POstGIS não estão prefixadas.

Topology support is build by default starting with PostGIS 2.0, and can be disabled specifying --without-topology configure option at build time as described in Chapter 2

8.1 Tipos de topologia

8.1.1 getfaceedges_returntype

getfaceedges_returntype — A composite type that consists of a sequence number and an edge number.

Descrição

A composite type that consists of a sequence number and an edge number. This is the return type for `ST_GetFaceEdges` and `GetNodeEdges` functions.

1. `sequence` é um inteiro: Refere-se a uma topologia definida na table topology.topology que define o esquema e srid da topologia.
2. `edge` é um inteiro: O identificador de um limite.

8.1.2 TopoGeometry

TopoGeometry — A composite type representing a topologically defined geometry.

Descrição

Um tipo composto que refere-se a uma geometria de topologia em uma camada específica da topologia, tendo um tipo e id específicos. Os elementos de uma TopoGeometry são as propriedades: topology_id, layer_id, inteireza id, inteireza do tipo.

1. `topology_id` é um inteiro: Refere-se a uma topologia definida na table topology.topology que define o esquema e srid da topologia.
2. `layer_id` é um inteiro: A layer_id nas layers tables que a TopoGEometry pertence. A combinação de topology_id, layer_id fornece uma referência única na table topology.layers.
3. `id` é um inteiro: a identidade é o número sequência autogerado que define a topogeometry na respectiva camada da topologia.
4. `type` inteiro entre 1 - 4 that define o tipo da geometria: 1:[multi]ponto, 2:[multi]linha, 3:[multi]poly, 4:coleção

Comportamento Casting

Esta seção lista os casts automáticos bem como os explícitos permitidos para esse tipo de dados

Cast To	Comportamento
geometria	automático

Veja também

CreateTopoGeom

8.1.3 validatetopology_returntype

validatetopology_returntype — A composite type that consists of an error message and id1 and id2 to denote location of error. This is the return type for `ValidateTopology`.

Descrição

Um tipo composto que consiste em uma mensagem de erro e dois inteiros. A função ValidateTopology retorna um conjunto para indicar erros de validação e a id1 e id2 para indicar as ids dos objetos da topologia envolvidas no erro.

1. `error` é varchar: Indica tipo de erro.

   A descrições de erro atuais são: nós coincidentes, limite cruza nó, limite não simples, geometria limite e nó que não combinam, limite começa e a geometria nó não combina, face sobrepõe face, face dentro de face,
2. `id1` é um inteiro: Indica identificador de limite / face / nós no erro.
3. `id2` é um inteiro: Para erros que envolvem limite / ou nó secundário

Veja também

ValidateTopology

8.2 Domínios de Topologia

8.2.1 TopoElement

TopoElement — Um arranjo de 2 inteiros geralmente usado para identificar um componente TopoGeometry.

Descrição

Um arranjo de 2 inteiros usados para representar um componente de um simples ou hierárquico TopoGeometry.

No caso de de uma TopoGeometria simples, o primeiro elemento do arranjo representa o identificador de um topológico primitivo, e o segundo elemento representa o tipo dele (1:nó, 2:limite, 3:face). No caso de uma TopoGeometria hierárquica o primeiro elemento do arranjo representa o identificador de uma TopoGeometria filha e o segundo elemento representa seu identificador de camada.

Note
Para qualquer uma das TopoGeometrias hierárquicas dadas, todos os elementos das TopoGeometrias filhas virão da mesma camada, assim com está especificado no relato topology.layer para a camada da TopoGeometria que está sendo definida.

Exemplos

```
SELECT te[1] AS id, te[2] AS type FROM
( SELECT ARRAY[1,2]::topology.topoelement AS te ) f;
 id | type
----+------
  1 |    2
```

```
SELECT ARRAY[1,2]::topology.topoelement;
  te
-------
 {1,2}
```

```
--Example of what happens when you try to case a 3 element array to topoelement
-- NOTE: topoement has to be a 2 element array so fails dimension check
SELECT ARRAY[1,2,3]::topology.topoelement;
ERROR:  value for domain topology.topoelement violates check constraint "dimensions"
```

Veja também

GetTopoGeomElements, TopoElementArray, TopoGeometry, TopoGeom_addElement, TopoGeom_remElement

8.2.2 TopoElementArray

TopoElementArray — An array of TopoElement objects.

Descrição

Um arranjo de 1 ou mais objetos TopoGeometria, geralmente usado para circular componentes dos objetos de TopoGeometrias.

Exemplos

```
SELECT '{{1,2},{4,3}}'::topology.topoelementarray As tea;
  tea
-------
{{1,2},{4,3}}

-- more verbose equivalent --
SELECT ARRAY[ARRAY[1,2], ARRAY[4,3]]::topology.topoelementarray As tea;

  tea
-------
{{1,2},{4,3}}

--using the array agg function packaged with topology --
SELECT topology.TopoElementArray_Agg(ARRAY[e,t]) As tea
  FROM generate_series(1,4) As e CROSS JOIN generate_series(1,3) As t;
  tea
------------------------------------------------------------------------
{{1,1},{1,2},{1,3},{2,1},{2,2},{2,3},{3,1},{3,2},{3,3},{4,1},{4,2},{4,3}}
```

```
SELECT '{{1,2,4},{3,4,5}}'::topology.topoelementarray As tea;
ERROR:  value for domain topology.topoelementarray violates check constraint "dimensions"
```

Veja também

TopoElement, GetTopoGeomElementArray, TopoElementArray_Agg

8.3 Gerenciamento de Topologia e TopoGeometria

8.3.1 AddTopoGeometryColumn

AddTopoGeometryColumn — Adiciona uma coluna topogeometria a uma table, registra essa coluna nova como uma camada topology.layer e retorna a nova layer_id.

Synopsis

integer **AddTopoGeometryColumn**(varchar topology_name, varchar schema_name, varchar table_name, varchar column_name, varchar feature_type);
integer **AddTopoGeometryColumn**(varchar topology_name, varchar schema_name, varchar table_name, varchar column_name, varchar feature_type, integer child_layer);

Descrição

Cada objeto TopoGeometria pertence à uma camada específica de uma Topologia específica. Antes de criar tal objeto, você precisa criar sua TopologyLayer. uma Camada de Topologia é uma associação de feature-table com a topologia. Também contém informações de tipo de hierárquicas. Nós criamos uma camada usando a função AddTopoGeometryColumn():

Esta função irá adicionar a coluna pedida e um relato para a table topology.layer com todas as informações dadas.

Se você não especificar [child_layer] (ou configurar para NULO) essa camada irá conter TopoGeometrias Básicas (compostas por elementos de topologia primitivos). Senão essa camada conterá TopoGeometrias hierárquicas (compostas por TopoGeometrias da child_layer).

Uma vez que a camada é criada (sua id retorna através da função AddTopoGeometryColumn) você pode construir objetos TopoGeometria nela.

Valid `feature_types` are: POINT, MULTIPOINT, LINE, MULTILINE, POLYGON, MULTIPOLYGON, COLLECTION

Availability: 1.1

Exemplos

```
-- Note for this example we created our new table in the ma_topo schema
-- though we could have created it in a different schema -- in which case topology_name and ↩
     schema_name would be different
CREATE SCHEMA ma;
CREATE TABLE ma.parcels(gid serial, parcel_id varchar(20) PRIMARY KEY, address text);
SELECT topology.AddTopoGeometryColumn('ma_topo', 'ma', 'parcels', 'topo', 'POLYGON');
```

```
CREATE SCHEMA ri;
CREATE TABLE ri.roads(gid serial PRIMARY KEY, road_name text);
SELECT topology.AddTopoGeometryColumn('ri_topo', 'ri', 'roads', 'topo', 'LINE');
```

Veja também

DropTopoGeometryColumn, toTopoGeom, Cria topologia, CreateTopoGeom

8.3.2 RenameTopoGeometryColumn

RenameTopoGeometryColumn — Renames a topogeometry column

Synopsis

topology.layer **RenameTopoGeometryColumn**(regclass layer_table, name feature_column, name new_name);

Descrição

This function changes the name of an existing TopoGeometry column ensuring metadata information about it is updated accordingly.

Availability: 3.4.0

Exemplos

```
SELECT topology.RenameTopoGeometryColumn('public.parcels', 'topogeom', 'tgeom');
```

Veja também

AddTopoGeometryColumn, RenameTopology

8.3.3 DropTopology

DropTopology — Cuidado ao usar: Derruba um esquema topologia e deleta sua referência da table topology.topology e referências para tables naquele esquema da table geometry_columns.

Synopsis

integer **DropTopology**(varchar topology_schema_name);

Descrição

Derruba um esquema topologia e deleta sua referência da table topology.topology e referências para tables naquele esquema da table geometry_columns.Esta função deve ser USADA COM CUIDADO, ela pode destruir algum dado importante. Se o esquema não existir. ela só remove entradas de referência do esquema nomeado.

Availability: 1.1

Exemplos

Cascata derruba o esquema ma_topo e remove todas as referências no topology.topology e geometry_columns.

```
SELECT topology.DropTopology('ma_topo');
```

Veja também

DropTopoGeometryColumn

8.3.4 RenameTopology

RenameTopology — Renames a topology

Synopsis

varchar **RenameTopology**(varchar old_name, varchar new_name);

Descrição

Renames a topology schema, updating its metadata record in the `topology.topology` table.

Availability: 3.4.0

Exemplos

Rename a topology from `topo_stage` to `topo_prod`.

```
SELECT topology.RenameTopology('topo_stage', 'topo_prod');
```

Veja também

CopyTopology, RenameTopoGeometryColumn

8.3.5 DropTopoGeometryColumn

DropTopoGeometryColumn — Derruba a coluna topogeometria da table nomeada `table_name` no esquema `schema_name` e tira os registros da

Synopsis

text **DropTopoGeometryColumn**(varchar schema_name, varchar table_name, varchar column_name);

Descrição

Derruba a coluna topogeometria da table nomeada `table_name` no esquema `schema_name` e tira os registros da colunas da table topology.layer. Retorna um resumo do drop status. NOTA: ela primeiro configura todos os valores para NULO antes de derrubar checks de integridade referencial.

Availability: 1.1

Exemplos

```
SELECT topology.DropTopoGeometryColumn('ma_topo', 'parcel_topo', 'topo');
```

Veja também

AddTopoGeometryColumn

8.3.6 Populate_Topology_Layer

Populate_Topology_Layer — Adds missing entries to topology.layer table by reading metadata from topo tables.

Synopsis

setof record **Populate_Topology_Layer**();

Descrição

Adds missing entries to the `topology.layer` table by inspecting topology constraints on tables. This function is useful for fixing up entries in topology catalog after restores of schemas with topo data.

It returns the list of entries created. Returned columns are `schema_name`, `table_name`, `feature_column`.

Disponibilidade: 2.3.0

Exemplos

```
SELECT CreateTopology('strk_topo');
CREATE SCHEMA strk;
CREATE TABLE strk.parcels(gid serial, parcel_id varchar(20) PRIMARY KEY, address text);
SELECT topology.AddTopoGeometryColumn('strk_topo', 'strk', 'parcels', 'topo', 'POLYGON');
-- this will return no records because this feature is already registered
SELECT *
  FROM topology.Populate_Topology_Layer();

-- let's rebuild
TRUNCATE TABLE topology.layer;

SELECT *
  FROM topology.Populate_Topology_Layer();

SELECT topology_id,layer_id, schema_name As sn, table_name As tn, feature_column As fc
FROM topology.layer;
```

```
schema_name | table_name | feature_column
-------------+------------+----------------
 strk        | parcels    | topo
(1 row)

 topology_id | layer_id |  sn  |   tn    |  fc
-------------+----------+------+---------+------
           2 |        2 | strk | parcels | topo
(1 row)
```

Veja também

AddTopoGeometryColumn

8.3.7 TopologySummary

TopologySummary — Takes a topology name and provides summary totals of types of objects in topology.

Synopsis

text **TopologySummary**(varchar topology_schema_name);

Descrição

Takes a topology name and provides summary totals of types of objects in topology.

Disponibilidade: 2.0.0

Exemplos

```
SELECT topology.topologysummary('city_data');
                   topologysummary
-----------------------------------------------------
 Topology city_data (329), SRID 4326, precision: 0
 22 nodes, 24 edges, 10 faces, 29 topogeoms in 5 layers
 Layer 1, type Polygonal (3), 9 topogeoms
  Deploy: features.land_parcels.feature
 Layer 2, type Puntal (1), 8 topogeoms
  Deploy: features.traffic_signs.feature
 Layer 3, type Lineal (2), 8 topogeoms
  Deploy: features.city_streets.feature
 Layer 4, type Polygonal (3), 3 topogeoms
  Hierarchy level 1, child layer 1
  Deploy: features.big_parcels.feature
 Layer 5, type Puntal (1), 1 topogeoms
  Hierarchy level 1, child layer 2
  Deploy: features.big_signs.feature
```

Veja também

Topology_Load_Tiger

8.3.8 ValidateTopology

ValidateTopology — Returns a set of validatetopology_returntype objects detailing issues with topology.

Synopsis

setof validatetopology_returntype **ValidateTopology**(varchar toponame, geometry bbox);

Descrição

Returns a set of validatetopology_returntype objects detailing issues with topology, optionally limiting the check to the area specified by the `bbox` parameter.

List of possible errors, what they mean and what the returned ids represent are displayed below:

Erro	id1	id2	Meaning
coincident nodes	Identifier of first node.	Identifier of second node.	Two nodes have the same geometry.
borda cruza nodo	Identifier of the edge.	Identifier of the node.	An edge has a node in its interior. See ST_Relate.
borda inválida	Identifier of the edge.		An edge geometry is invalid. See ST_IsValid.
borda não simples	Identifier of the edge.		An edge geometry has self-intersections. See ST_IsSimple.
borda cruza borda	Identifier of first edge.	Identifier of second edge.	Two edges have an interior intersection. See ST_Relate.
edge start node geometry mis-match	Identifier of the edge.	Identifier of the indicated start node.	The geometry of the node indicated as the starting node for an edge does not match the first point of the edge geometry. See ST_StartPoint.
edge end node geometry mis-match	Identifier of the edge.	Identifier of the indicated end node.	The geometry of the node indicated as the ending node for an edge does not match the last point of the edge geometry. See ST_EndPoint.
face sem bordas	Identifier of the orphaned face.		No edge reports an existing face on either of its sides (left_face, right_face).
face sem anéis	Identifier of the partially-defined face.		Edges reporting a face on their sides do not form a ring.
face has wrong mbr	Identifier of the face with wrong mbr cache.		Minimum bounding rectangle of a face does not match minimum bounding box of the collection of edges reporting the face on their sides.
hole not in advertised face	Signed identifier of an edge, identifying the ring. See GetRingEdges.		A ring of edges reporting a face on its exterior is contained in different face.

Erro	id1	id2	Meaning
not-isolated node has not-containing_face	Identifier of the ill-defined node.		A node which is reported as being on the boundary of one or more edges is indicating a containing face.
isolated node has containing_face	Identifier of the ill-defined node.		A node which is not reported as being on the boundary of any edges is lacking the indication of a containing face.
isolated node has wrong containing_face	Identifier of the misrepresented node.		A node which is not reported as being on the boundary of any edges indicates a containing face which is not the actual face containing it. See GetFaceContainingPoint.
invalid next_right_edge	Identifier of the misrepresented edge.	Signed id of the edge which should be indicated as the next right edge.	The edge indicated as the next edge encountered walking on the right side of an edge is wrong.
invalid next_left_edge	Identifier of the misrepresented edge.	Signed id of the edge which should be indicated as the next left edge.	The edge indicated as the next edge encountered walking on the left side of an edge is wrong.
mixed face labeling in ring	Signed identifier of an edge, identifying the ring. See GetRingEdges.		Edges in a ring indicate conflicting faces on the walking side. This is also known as a "Side Location Conflict".
non-closed ring	Signed identifier of an edge, identifying the ring. See GetRingEdges.		A ring of edges formed by following next_left_edge/next_right_edge attributes starts and ends on different nodes.
face has multiple shells	Identifier of the contended face.	Signed identifier of an edge, identifying the ring. See GetRingEdges.	More than a one ring of edges indicate the same face on its interior.

Disponibilidade: 1.0.0

Melhorias: 2.0.0 limite mais eficiente cruzando detenção e consertos para falsos positivos que existiam em versões anteriores.

Alterações: 2.2.0 valores para id1 e id2 foram trocados para "limite cruza nó", para serem consistentes com a descrição do erro.

Changed: 3.2.0 added optional bbox parameter, perform face labeling and edge linking checks.

Exemplos

```
SELECT * FROM  topology.ValidateTopology('ma_topo');
      error         | id1 | id2
--------------------+-----+-----
face without edges |   1 |
```

Veja também

validatetopology_returntype, Topology_Load_Tiger

8.3.9 ValidateTopologyRelation

ValidateTopologyRelation — Returns info about invalid topology relation records

Synopsis

setof record **ValidateTopologyRelation**(varchar toponame);

Descrição

Returns a set records giving information about invalidities in the relation table of the topology.

Availability: 3.2.0

Veja também

ValidateTopology

8.3.10 FindTopology

FindTopology — Returns a topology record by different means.

Synopsis

topology **FindTopology**(TopoGeometry topogeom);
topology **FindTopology**(regclass layerTable, name layerColumn);
topology **FindTopology**(name layerSchema, name layerTable, name layerColumn);
topology **FindTopology**(text topoName);
topology **FindTopology**(int id);

Descrição

Takes a topology identifier or the identifier of a topology-related object and returns a topology.topology record.

Availability: 3.2.0

Exemplos

```
SELECT name(findTopology('features.land_parcels', 'feature'));
   name
-----------
 city_data
(1 row)
```

Veja também

FindLayer

8.3.11 FindLayer

FindLayer — Returns a topology.layer record by different means.

Synopsis

topology.layer **FindLayer**(TopoGeometry tg);
topology.layer **FindLayer**(regclass layer_table, name feature_column);
topology.layer **FindLayer**(name schema_name, name table_name, name feature_column);
topology.layer **FindLayer**(integer topology_id, integer layer_id);

Descrição

Takes a layer identifier or the identifier of a topology-related object and returns a topology.layer record.

Availability: 3.2.0

Exemplos

```
SELECT layer_id(findLayer('features.land_parcels', 'feature'));
 layer_id
----------
        1
(1 row)
```

Veja também

FindTopology

8.4 Topology Statistics Management

Adding elements to a topology triggers many database queries for finding existing edges that will be split, adding nodes and updating edges that will node with the new linework. For this reason it is useful that statistics about the data in the topology tables are up-to-date.

PostGIS Topology population and editing functions do not automatically update the statistics because a updating stats after each and every change in a topology would be overkill, so it is the caller's duty to take care of that.

Note
That the statistics updated by autovacuum will NOT be visible to transactions which started before autovacuum process completed, so long-running transactions will need to run ANALYZE themselves, to use updated statistics.

8.5 Construtores de topologia

8.5.1 Cria topologia

Cria topologia — Creates a new topology schema and registers it in the topology.topology table.

Synopsis

integer **CreateTopology**(varchar topology_schema_name);
integer **CreateTopology**(varchar topology_schema_name, integer srid);
integer **CreateTopology**(varchar topology_schema_name, integer srid, double precision prec);
integer **CreateTopology**(varchar topology_schema_name, integer srid, double precision prec, boolean hasz);

Descrição

Creates a new topology schema with name `topology_name` and registers it in the `topology.topology` table. Topologies must be uniquely named. The topology tables (`edge_data`, `face`, `node`,and `relation` are created in the schema. It returns the id of the topology.

The `srid` is the spatial reference system SRID for the topology.

The tolerance `prec` is measured in the units of the spatial reference system. The tolerance defaults to 0.

`hasz` defaults to false if not specified.

This is similar to the SQL/MM ST_InitTopoGeo but has more functionality.

Availability: 1.1

Enhanced: 2.0 added the signature accepting hasZ

Exemplos

Create a topology schema called `ma_topo` that stores edges and nodes in Massachusetts State Plane-meters (SRID = 26986). The tolerance represents 0.5 meters since the spatial reference system is meter-based.

```
SELECT topology.CreateTopology('ma_topo', 26986, 0.5);
```

Create a topology for Rhode Island called `ri_topo` in spatial reference system State Plane-feet (SRID = 3438)

```
SELECT topology.CreateTopology('ri_topo', 3438) AS topoid;
topoid
------
2
```

Veja também

Section 4.5, ST_InitTopoGeo, Topology_Load_Tiger

8.5.2 CopyTopology

CopyTopology — Makes a copy of a topology (nodes, edges, faces, layers and TopoGeometries) into a new schema

Synopsis

integer **CopyTopology**(varchar existing_topology_name, varchar new_name);

Descrição

Creates a new topology with name `new_name`, with SRID and precision copied from `existing_topology_name` The nodes, edges and faces in `existing_topology_name` are copied into the new topology, as well as Layers and their associated TopoGeometries.

Note
The new rows in the `topology.layer` table contain synthetic values for schema_name, table_name and feature_column. This is because the TopoGeometry objects exist only as a definition and are not yet available in a user-defined table.

Disponibilidade: 2.0.0

Exemplos

Make a backup of a topology called `ma_topo`.

```
SELECT topology.CopyTopology('ma_topo', 'ma_topo_backup');
```

Veja também

Section 4.5, Cria topologia, RenameTopology

8.5.3 ST_InitTopoGeo

ST_InitTopoGeo — Creates a new topology schema and registers it in the topology.topology table.

Synopsis

text **ST_InitTopoGeo**(varchar topology_schema_name);

Descrição

This is the SQL-MM equivalent of Cria topologia. It lacks options for spatial reference system and tolerance. it returns a text description of the topology creation, instead of the topology id.

Availability: 1.1

 This method implements the SQL/MM specification. SQL-MM 3 Topo-Geo and Topo-Net 3: Routine Details: X.3.17

Exemplos

```
SELECT topology.ST_InitTopoGeo('topo_schema_to_create') AS topocreation;
                      astopocreation
-----------------------------------------------------------
 Topology-Geometry 'topo_schema_to_create' (id:7) created.
```

Veja também

Cria topologia

8.5.4 ST_CreateTopoGeo

ST_CreateTopoGeo — Adiciona uma coleção de geometrias para uma dada topologia vazia e retorna uma mensagem detalhando sucesso.

Synopsis

text **ST_CreateTopoGeo**(varchar atopology, geometry acollection);

Descrição

Adiciona uma coleção de geometrias para uma dada topologia vazia e retorna uma mensagem detalhando sucesso.

Útil para popular uma topologia vazia.

Disponibilidade: 2.0

 This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details -- X.3.18

Exemplos

```
-- Populate topology --
SELECT topology.ST_CreateTopoGeo('ri_topo',
 ST_GeomFromText('MULTILINESTRING((384744 236928,384750 236923,384769 236911,384799  ↩
    236895,384811 236890,384833 236884,
  384844 236882,384866 236881,384879 236883,384954 236898,385087 236932,385117 236938,
  385167 236938,385203 236941,385224 236946,385233 236950,385241 236956,385254 236971,
  385260 236979,385268 236999,385273 237018,385273 237037,385271 237047,385267 237057,
  385225 237125,385210 237144,385192 237161,385167 237192,385162 237202,385159 237214,
  385159 237227,385162 237241,385166 237256,385196 237324,385209 237345,385234 237375,
  385237 237383,385238 237399,385236 237407,385227 237419,385213 237430,385193 237439,
  385174 237451,385170 237455,385169 237460,385171 237475,385181 237503,385190 237521,
  385200 237533,385206 237538,385213 237541,385221 237542,385235 237540,385242 237541,
  385249 237544,385260 237555,385270 237570,385289 237584,385292 237589,385291  ↩
    237596,385284 237630))',3438)
  );

     st_createtopogeo
----------------------------
 Topology ri_topo populated


-- create tables and topo geometries --
CREATE TABLE ri.roads(gid serial PRIMARY KEY, road_name text);

SELECT topology.AddTopoGeometryColumn('ri_topo', 'ri', 'roads', 'topo', 'LINE');
```

Veja também

AddTopoGeometryColumn, Cria topologia, DropTopology

8.5.5 TopoGeo_AddPoint

TopoGeo_AddPoint — Adiciona um ponto a uma topologia usando uma tolerância e possivelmente dividindo um limite existente.

Synopsis

integer **TopoGeo_AddPoint**(varchar atopology, geometry apoint, float8 tolerance);

Descrição

Adds a point to an existing topology and returns its identifier. The given point will snap to existing nodes or edges within given tolerance. An existing edge may be split by the snapped point.

Disponibilidade: 2.0.0

Veja também

TopoGeo_AddLineString, TopoGeo_AddPolygon, AddNode, Cria topologia

8.5.6 TopoGeo_AddLineString

TopoGeo_AddLineString — Adds a linestring to an existing topology using a tolerance and possibly splitting existing edges/-faces. Returns edge identifiers.

Synopsis

SETOF integer **TopoGeo_AddLineString**(varchar atopology, geometry aline, float8 tolerance);

Descrição

Adds a linestring to an existing topology and returns a set of edge identifiers forming it up. The given line will snap to existing nodes or edges within given tolerance. Existing edges and faces may be split by the line.

Note
Updating statistics about topologies being loaded via this function is up to caller, see maintaining statistics during topology editing and population.

Disponibilidade: 2.0.0

Veja também

TopoGeo_AddPoint, TopoGeo_AddPolygon, AddEdge, Cria topologia

8.5.7 TopoGeo_AddPolygon

TopoGeo_AddPolygon — Adds a polygon to an existing topology using a tolerance and possibly splitting existing edges/faces. Returns face identifiers.

Synopsis

SETOF integer **TopoGeo_AddPolygon**(varchar atopology, geometry apoly, float8 tolerance);

Descrição

Adds a polygon to an existing topology and returns a set of face identifiers forming it up. The boundary of the given polygon will snap to existing nodes or edges within given tolerance. Existing edges and faces may be split by the boundary of the new polygon.

Note
Updating statistics about topologies being loaded via this function is up to caller, see maintaining statistics during topology editing and population.

Disponibilidade: 2.0.0

Veja também

TopoGeo_AddPoint, TopoGeo_AddLineString, AddFace, Cria topologia

8.6 Editores de Topologia

8.6.1 ST_AddIsoNode

ST_AddIsoNode — Adiciona um nó isolado a uma face em uma topologia e retorna a id do novo nó. Se a face é nula, o nó continua sendo criado.

Synopsis

integer **ST_AddIsoNode**(varchar atopology, integer aface, geometry apoint);

Descrição

Adiciona um nó isolado com a localização do ponto `apoint` com uma face existente com faceid `aface` a uma topologia `atopology` e retorna a nodeid do novo nó.

O sistema de referência espacial (srid) da geometria pontual não é o mesmo que a topologia, o `apoint` não é uma geometria pontual, o ponto é nulo, ou o ponto intersecta um limite existente (mesmo nos limites), então uma exceção é aberta. Se o ponto já existe como um nó, uma exceção é aberta.

Se `aface` não é nula e o `apoint` não está dentro da face, então, uma exceção é aberta.

Availability: 1.1

 This method implements the SQL/MM specification. SQL-MM: Topo-Net Routines: X+1.3.1

Exemplos

Veja também

AddNode, Cria topologia, DropTopology, ST_Intersects

8.6.2 ST_AddIsoEdge

ST_AddIsoEdge — Adiciona um limite isolado definido pela geometria `alinestring` a uma topologia conectando dois nós isoladosanode e `anothernode` e retorna a nova id do novo limite.

Synopsis

integer **ST_AddIsoEdge**(varchar atopology, integer anode, integer anothernode, geometry alinestring);

Descrição

Adiciona um limite isolado definido pela geometria `alinestring` a uma topologia conectando dois nós isoladosanode e `anothernode` e retorna a nova id do novo limite.

Se o sistema de referência espacial (srid) da geometria `alinestring` não for o mesmo da topologia, qualquer argumento de entrada é nulo, ou is nós estão contidos em mais de uma face, ou eles são o começo ou fim de um limite existente, então, uma exceção é aberta.

Se a `alinestring` não está dentro da face da face o `anode` e `anothernode` pertence, então, uma exceção é aberta.

Se o `anode` e `anothernode` não são os pontos de começo e fim da `alinestring` então, uma exceção é aberta.

Availability: 1.1

 This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.4

Exemplos

Veja também

ST_AddIsoNode, ST_IsSimple, ST_Within

8.6.3 ST_AddEdgeNewFaces

ST_AddEdgeNewFaces — Adiciona um novo limite e, se uma face for dividida, deleta a face original e substitui por duas novas faces.

Synopsis

integer **ST_AddEdgeNewFaces**(varchar atopology, integer anode, integer anothernode, geometry acurve);

Descrição

Adiciona um novo limite e, se uma face for dividida, deleta a face original e substitui por duas novas faces. Retorna a id do novo limite adicionado.

Atualiza todos os limites existentes e relacionamentos em conformidade.

Se algum argumento for nulo, os nós são desconhecidos (devem existir na table `node` do esquema de topologia), a `acurve` não é uma `LINESTRING`, o `anode` e `anothernode` não são os pontos de começo e fim da `acurve`, logo, um erro é lançado.

Se o sistema de referência espacial (srid) da geometria `acurve` não for o mesmo da topologia, uma exceção é lançada.

Disponibilidade: 2.0

 This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.12

Exemplos

Veja também

ST_RemEdgeNewFace

ST_AddEdgeModFace

8.6.4 ST_AddEdgeModFace

ST_AddEdgeModFace — Adiciona um novo limite e, se uma face for dividida, modifica a face original e adiciona uma nova face.

Synopsis

integer **ST_AddEdgeModFace**(varchar atopology, integer anode, integer anothernode, geometry acurve);

Descrição

Adiciona um novo limite e, se uma face for dividida, modifica a face original e adiciona uma nova.

Note
Se possível, a face nova será criada no lado esquerdo do novo limite. Isto não será possível se a face do lado esquerdo precisar ser a face universal (sem limites).

Retorna a id do novo limite adicionado.

Atualiza todos os limites existentes e relacionamentos em conformidade.

Se algum argumento for nulo, os nós são desconhecidos (devem existir na table `node` do esquema de topologia), a `acurve` não é uma `LINESTRING`, o `anode` e `anothernode` não são os pontos de começo e fim da `acurve`, logo, um erro é lançado.

Se o sistema de referência espacial (srid) da geometria `acurve` não for o mesmo da topologia, uma exceção é lançada.

Disponibilidade: 2.0

This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.13

Exemplos

Veja também

ST_RemEdgeModFace

ST_AddEdgeNewFaces

8.6.5 ST_RemEdgeNewFace

ST_RemEdgeNewFace — Remove um limite e, se o limite removido separava duas faces, deleta as faces originais e as substitui por uma nova face.

Synopsis

integer **ST_RemEdgeNewFace**(varchar atopology, integer anedge);

Descrição

Remove um limite e, se o limite removido separava duas faces, deleta as faces originais e as substitui por uma nova face.

Retorna a id de uma face nova criada ou NULA, se nenhuma face nova for criada. Nenhuma face nova é criada quando o limite removido está pendurado, isolado ou confinado na face universal (possivelmente fazendo a inundação universal dentro da face no outro lado).

Atualiza todos os limites existentes e relacionamentos em conformidade.

Refuses to remove an edge participating in the definition of an existing TopoGeometry. Refuses to heal two faces if any TopoGeometry is defined by only one of them (and not the other).

Se qualquer argumento for nulo, o limite dado é desconhecido (deve existir na table `edge` do esquema de topologia), o nome da topologia é inválido, logo, um erro é lançado.

Disponibilidade: 2.0

This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.14

Exemplos

Veja também

ST_RemEdgeModFace

ST_AddEdgeNewFaces

8.6.6 ST_RemEdgeModFace

ST_RemEdgeModFace — Removes an edge, and if the edge separates two faces deletes one face and modifies the other face to cover the space of both.

Synopsis

integer **ST_RemEdgeModFace**(varchar atopology, integer anedge);

Descrição

Removes an edge, and if the removed edge separates two faces deletes one face and modifies the other face to cover the space of both. Preferentially keeps the face on the right, to be consistent with ST_AddEdgeModFace. Returns the id of the face which is preserved.

Atualiza todos os limites existentes e relacionamentos em conformidade.

Refuses to remove an edge participating in the definition of an existing TopoGeometry. Refuses to heal two faces if any TopoGeometry is defined by only one of them (and not the other).

Se qualquer argumento for nulo, o limite dado é desconhecido (deve existir na table `edge` do esquema de topologia), o nome da topologia é inválido, logo, um erro é lançado.

Disponibilidade: 2.0

 This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.15

Exemplos

Veja também

ST_AddEdgeModFace

ST_RemEdgeNewFace

8.6.7 ST_ChangeEdgeGeom

ST_ChangeEdgeGeom — Modifica a forma de um limite sem afetar a estrutura da topologia.

Synopsis

integer **ST_ChangeEdgeGeom**(varchar atopology, integer anedge, geometry acurve);

Descrição

Modifica a forma de um limite sem afetar a estrutura da topologia.

If any arguments are null, the given edge does not exist in the `edge` table of the topology schema, the `acurve` is not a `LINESTRING`, or the modification would change the underlying topology then an error is thrown.

Se o sistema de referência espacial (srid) da geometria `acurve` não for o mesmo da topologia, uma exceção é lançada.

Se a nova `acurve` não for simples, um erro é lançado.

Se mover o limite de uma posição antiga acertar um obstáculo, um erro é lançado.

Disponibilidade: 1.1.0

Melhorias: 2.0.0 adiciona execução da consistência topológica

 This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details X.3.6

Exemplos

```
SELECT topology.ST_ChangeEdgeGeom('ma_topo', 1,
                ST_GeomFromText('LINESTRING(227591.9 893900.4,227622.6 893844.3,227641.6  ↩
                    893816.6, 227704.5 893778.5)', 26986) );
 ----
 Edge 1 changed
```

Veja também

ST_AddEdgeModFace

ST_RemEdgeModFace

ST_ModEdgeSplit

8.6.8 ST_ModEdgeSplit

ST_ModEdgeSplit — Divide um limite criando um novo nó junto de um limite existente, modificando o limite original e adicionando um novo limite.

Synopsis

integer **ST_ModEdgeSplit**(varchar atopology, integer anedge, geometry apoint);

Descrição

Divide um limite criando um novo nó junto de um limite existente, modificando o limite original e adicionando um novo limite. Atualiza todos os limites e relacionamentos em conformidade. Retorna o identificador do novo nó adicionado.

Availability: 1.1

Alterações: 2.0 - Nas versões anteriores, isto recebia o nome errado ST_ModEdgesSplit

 This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.9

Exemplos

```
-- Add an edge --
 SELECT topology.AddEdge('ma_topo', ST_GeomFromText('LINESTRING(227592 893910, 227600  ↩
     893910)', 26986) ) As edgeid;

-- edgeid-
3


-- Split the edge  --
SELECT topology.ST_ModEdgeSplit('ma_topo',  3, ST_SetSRID(ST_Point(227594,893910),26986)  ) ↩
     As node_id;
        node_id
-------------------------
7
```

Veja também

ST_NewEdgesSplit, ST_ModEdgeHeal, ST_NewEdgeHeal, AddEdge

8.6.9 ST_ModEdgeHeal

ST_ModEdgeHeal — Heals two edges by deleting the node connecting them, modifying the first edge and deleting the second edge. Returns the id of the deleted node.

Synopsis

int **ST_ModEdgeHeal**(varchar atopology, integer anedge, integer anotheredge);

Descrição

Heals two edges by deleting the node connecting them, modifying the first edge and deleting the second edge. Returns the id of the deleted node. Updates all existing joined edges and relationships accordingly.

Disponibilidade: 2.0

 This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.9

Veja também

ST_ModEdgeSplit ST_NewEdgesSplit

8.6.10 ST_NewEdgeHeal

ST_NewEdgeHeal — Heals two edges by deleting the node connecting them, deleting both edges, and replacing them with an edge whose direction is the same as the first edge provided.

Synopsis

int **ST_NewEdgeHeal**(varchar atopology, integer anedge, integer anotheredge);

Descrição

Heals two edges by deleting the node connecting them, deleting both edges, and replacing them with an edge whose direction is the same as the first edge provided. Returns the id of the new edge replacing the healed ones. Updates all existing joined edges and relationships accordingly.

Disponibilidade: 2.0

 This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.9

Veja também

ST_ModEdgeHeal ST_ModEdgeSplit ST_NewEdgesSplit

8.6.11 ST_MoveIsoNode

ST_MoveIsoNode — Moves an isolated node in a topology from one point to another. If new `apoint` geometry exists as a node an error is thrown. Returns description of move.

Synopsis

text **ST_MoveIsoNode**(varchar atopology, integer anode, geometry apoint);

Descrição

Move um nó isolado em uma topologia de um ponto para outro. Se nova geometria `apoint` existe como um nó, um erro é lançado.

If any arguments are null, the `apoint` is not a point, the existing node is not isolated (is a start or end point of an existing edge), new node location intersects an existing edge (even at the end points) or the new location is in a different face (since 3.2.0) then an exception is thrown.

Se o sistema de referência espacial (srid) da geometria pontual não for o mesmo da topologia, uma exceção é lançada.

Disponibilidade: 2.0.0

Enhanced: 3.2.0 ensures the nod cannot be moved in a different face

 This method implements the SQL/MM specification. SQL-MM: Topo-Net Routines: X.3.2

Exemplos

```
-- Add an isolated node with no face  --
SELECT topology.ST_AddIsoNode('ma_topo',  NULL, ST_GeomFromText('POINT(227579 893916)',  ↩
    26986) ) As nodeid;
 nodeid
--------
      7
-- Move the new node --
SELECT topology.ST_MoveIsoNode('ma_topo', 7,  ST_GeomFromText('POINT(227579.5 893916.5)',  ↩
    26986) ) As descrip;
                      descrip
-------------------------------------------------
Isolated Node 7 moved to location 227579.5,893916.5
```

Veja também

ST_AddIsoNode

8.6.12 ST_NewEdgesSplit

ST_NewEdgesSplit — Divide um limite criando um novo nó ao longo do limite existente, deletando o limite original e substituindo-o por dois novos. Retorna a id do novo nó criado que integra os novos limites.

Synopsis

integer **ST_NewEdgesSplit**(varchar atopology, integer anedge, geometry apoint);

Descrição

Divide um limite com uma id limite `anedge` criando um novo nó com uma localização de ponto `apoint` junto co i limite atual, deletando o limite original e substituindo-o por dois novos. Retorna a id do novo nó criado que se une aos novos limites. Atualiza todos os limites unidos e relacionamentos em conformidade.

Se o sistema de referência espacial (srid) da geometria pontual não é o mesmo que a topologia, o `apoint` não é uma geometria pontual, o ponto é nulo, o ponto já existe como um nó, o limite não corresponde a um limite existente ou o ponto não está dentro do limite, então, uma exceção é aberta.

Availability: 1.1

 This method implements the SQL/MM specification. SQL-MM: Topo-Net Routines: X.3.8

Exemplos

```
-- Add an edge  --
SELECT topology.AddEdge('ma_topo', ST_GeomFromText('LINESTRING(227575 893917,227592 893900) ↩
    ', 26986) ) As edgeid;
-- result-
edgeid
------
        2
-- Split the new edge --
SELECT topology.ST_NewEdgesSplit('ma_topo', 2,  ST_GeomFromText('POINT(227578.5 893913.5)', ↩
     26986) ) As newnodeid;
 newnodeid
---------
       6
```

Veja também

ST_ModEdgeSplit ST_ModEdgeHeal ST_NewEdgeHeal AddEdge

8.6.13 ST_RemoveIsoNode

ST_RemoveIsoNode — Remove um nó isolado e retorna descrição de ação. Se o nó não for isolado (for começo ou fim de um limite), então, uma exceção é lançada.

Synopsis

text **ST_RemoveIsoNode**(varchar atopology, integer anode);

Descrição

Remove um nó isolado e retorna descrição de ação. Se o nó não for isolado (for começo ou fim de um limite), então, uma exceção é lançada.

Availability: 1.1

 This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X+1.3.3

Exemplos

```
-- Remove an isolated node with no face  --
SELECT topology.ST_RemoveIsoNode('ma_topo',  7 ) As result;
         result
-------------------------
 Isolated node 7 removed
```

Veja também

ST_AddIsoNode

8.6.14 ST_RemoveIsoEdge

ST_RemoveIsoEdge — Removes an isolated edge and returns description of action. If the edge is not isolated, then an exception is thrown.

Synopsis

text **ST_RemoveIsoEdge**(varchar atopology, integer anedge);

Descrição

Removes an isolated edge and returns description of action. If the edge is not isolated, then an exception is thrown.

Availability: 1.1

 This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X+1.3.3

Exemplos

```
-- Remove an isolated node with no face  --
SELECT topology.ST_RemoveIsoNode('ma_topo',  7 ) As result;
         result
-------------------------
 Isolated node 7 removed
```

Veja também

ST_AddIsoNode

8.7 Assessores de Topologia

8.7.1 GetEdgeByPoint

GetEdgeByPoint — Finds the edge-id of an edge that intersects a given point.

Synopsis

integer **GetEdgeByPoint**(varchar atopology, geometry apoint, float8 tol1);

Descrição

Retrieves the id of an edge that intersects a Point.

A função retorna uma inteireza (id-limite) dada uma topologia, um PONTO e uma tolerância. Se tolerância = 0, o ponto tem que intersectar o limite.

If `apoint` doesn't intersect an edge, returns 0 (zero).

Se usa tolerância > 0 e não existe mais que um limite próximo ao ponto, uma exceção é lançada.

Note
Se tolerância = 0, a função usa ST_Intersects, senão usa ST_DWithin.

Desempenhado pelo módulo GEOS.

Disponibilidade: 2.0.0

Exemplos

Estes exemplos utilizam limites que criamos em

```
SELECT topology.GetEdgeByPoint('ma_topo',geom, 1) As with1mtol, topology.GetEdgeByPoint(' ↩
    ma_topo',geom,0) As withnotol
FROM ST_GeomFromEWKT('SRID=26986;POINT(227622.6 893843)') As geom;
 with1mtol | withnotol
-----------+-----------
         2 |         0
```

```
SELECT topology.GetEdgeByPoint('ma_topo',geom, 1) As nearnode
FROM ST_GeomFromEWKT('SRID=26986;POINT(227591.9 893900.4)') As geom;

-- get error --
ERROR:  Two or more edges found
```

Veja também

AddEdge, GetNodeByPoint, GetFaceByPoint

8.7.2 GetFaceByPoint

GetFaceByPoint — Finds face intersecting a given point.

Synopsis

integer **GetFaceByPoint**(varchar atopology, geometry apoint, float8 tol1);

Descrição

Finds a face referenced by a Point, with given tolerance.

The function will effectively look for a face intersecting a circle having the point as center and the tolerance as radius.

If no face intersects the given query location, 0 is returned (universal face).

If more than one face intersect the query location an exception is thrown.

Disponibilidade: 2.0.0

Enhanced: 3.2.0 more efficient implementation and clearer contract, stops working with invalid topologies.

Exemplos

```
SELECT topology.GetFaceByPoint('ma_topo',geom, 10) As with1mtol, topology.GetFaceByPoint(' ↩
    ma_topo',geom,0) As withnotol
        FROM ST_GeomFromEWKT('POINT(234604.6 899382.0)') As geom;

         with1mtol | withnotol
        -----------+-----------
                         1 |         0
```

```
SELECT topology.GetFaceByPoint('ma_topo',geom, 1) As nearnode
        FROM ST_GeomFromEWKT('POINT(227591.9 893900.4)') As geom;

-- get error --
ERROR:  Two or more faces found
```

Veja também

GetFaceContainingPoint, AddFace, GetNodeByPoint, GetEdgeByPoint

8.7.3 GetFaceContainingPoint

GetFaceContainingPoint — Finds the face containing a point.

Synopsis

integer **GetFaceContainingPoint**(text atopology, geometry apoint);

Descrição

Returns the id of the face containing a point.

An exception is thrown if the point falls on a face boundary.

Note
The function relies on a valid topology, using edge linking and face labeling.

Availability: 3.2.0

Veja também

ST_GetFaceGeometry

8.7.4 GetNodeByPoint

GetNodeByPoint — Finds the node-id of a node at a point location.

Synopsis

integer **GetNodeByPoint**(varchar atopology, geometry apoint, float8 tol1);

Descrição

Retrieves the id of a node at a point location.

The function returns an integer (id-node) given a topology, a POINT and a tolerance. If tolerance = 0 means exact intersection, otherwise retrieves the node from an interval.

If `apoint` doesn't intersect a node, returns 0 (zero).

If use tolerance > 0 and there is more than one node near the point then an exception is thrown.

Note
Se tolerância = 0, a função usa ST_Intersects, senão usa ST_DWithin.

Desempenhado pelo módulo GEOS.

Disponibilidade: 2.0.0

Exemplos

Estes exemplos utilizam limites que criamos em

```
SELECT topology.GetNodeByPoint('ma_topo',geom, 1) As nearnode
 FROM ST_GeomFromEWKT('SRID=26986;POINT(227591.9 893900.4)') As geom;
  nearnode
----------
         2
```

```
SELECT topology.GetNodeByPoint('ma_topo',geom, 1000) As too_much_tolerance
 FROM ST_GeomFromEWKT('SRID=26986;POINT(227591.9 893900.4)') As geom;

 ----get error--
 ERROR:  Two or more nodes found
```

Veja também

AddEdge, GetEdgeByPoint, GetFaceByPoint

8.7.5 GetTopologyID

GetTopologyID — Retorna a id de uma topologia na table topology.topology dado o nome da topologia.

Synopsis

integer **GetTopologyID**(varchar toponame);

Descrição

Retorna a id de uma topologia na table topology.topology dado o nome da topologia.

Availability: 1.1

Exemplos

```
SELECT topology.GetTopologyID('ma_topo') As topo_id;
 topo_id
---------
       1
```

Veja também

Cria topologia, DropTopology, GetTopologyName, GetTopologySRID

8.7.6 GetTopologySRID

GetTopologySRID — Retorna o SRID de uma topologia na table topology.topology dado o nome da topologia.

Synopsis

integer **GetTopologyID**(varchar toponame);

Descrição

Retorna a id de referência espacial de uma topologia na table topology.topology dado o nome da topologia.

Disponibilidade: 2.0.0

Exemplos

```
SELECT topology.GetTopologySRID('ma_topo') As SRID;
 SRID
-------
  4326
```

Veja também

Cria topologia, DropTopology, GetTopologyName, GetTopologyID

8.7.7 GetTopologyName

GetTopologyName — Retorna o nome de uma topologia (esquema) dada a id da topologia.

Synopsis

varchar **GetTopologyName**(integer topology_id);

Descrição

Retorna o nome da topologia (esquema) de uma table topology.topology dada a id topologia dela.

Availability: 1.1

Exemplos

```
SELECT topology.GetTopologyName(1) As topo_name;
 topo_name
-----------
 ma_topo
```

Veja também

Cria topologia, DropTopology, GetTopologyID, GetTopologySRID

8.7.8 ST_GetFaceEdges

ST_GetFaceEdges — Retorna um conjunto de limites ordenados que amarram `aface`.

Synopsis

getfaceedges_returntype **ST_GetFaceEdges**(varchar atopology, integer aface);

Descrição

Retorna um conjunto de limites ordenados que amarram `aface`. Cada saída consiste em uma sequência e uma limiteid. Os números das sequências começam com o valor 1.

A enumeração dos limites de cada anel começa do limite com o menos identificador. A ordem de limites segue uma regra da mão esquerda (a face amarrada está a esquerda de cada limite direto).

Disponibilidade: 2.0

 This method implements the SQL/MM specification. SQL-MM 3 Topo-Geo and Topo-Net 3: Routine Details: X.3.5

Exemplos

```
-- Returns the edges bounding face 1
SELECT (topology.ST_GetFaceEdges('tt', 1)).*;
-- result --
 sequence | edge
----------+------
        1 |   -4
        2 |    5
        3 |    7
        4 |   -6
        5 |    1
        6 |    2
        7 |    3
(7 rows)
```

```
-- Returns the sequence, edge id
-- and geometry of the edges that bound face 1
-- If you just need geom and seq, can use ST_GetFaceGeometry
SELECT t.seq, t.edge, geom
FROM topology.ST_GetFaceEdges('tt',1) As t(seq,edge)
        INNER JOIN tt.edge AS e ON abs(t.edge) = e.edge_id;
```

Veja também

GetRingEdges, AddFace, ST_GetFaceGeometry

8.7.9 ST_GetFaceGeometry

ST_GetFaceGeometry — Retorna o polígono na topologia dada com a id de face especificada.

Synopsis

geometry **ST_GetFaceGeometry**(varchar atopology, integer aface);

Descrição

Retorna o polígono na topologia dada com a id de face especificada. Constrói o polígono dos limites fazendo a face.

Availability: 1.1

 This method implements the SQL/MM specification. SQL-MM 3 Topo-Geo and Topo-Net 3: Routine Details: X.3.16

Exemplos

```
-- Returns the wkt of the polygon added with AddFace
SELECT ST_AsText(topology.ST_GetFaceGeometry('ma_topo', 1)) As facegeomwkt;
-- result --
               facegeomwkt

------------------------------------------------------------------------------
 POLYGON((234776.9 899563.7,234896.5 899456.7,234914 899436.4,234946.6 899356.9,
234872.5 899328.7,234891 899285.4,234992.5 899145,234890.6 899069,
234755.2 899255.4,234612.7 899379.4,234776.9 899563.7))
```

Veja também

AddFace

8.7.10 GetRingEdges

GetRingEdges — Retorna o conjunto ordenado de identificadores de limites assinados, conhecidos caminhando em um lado da beirada.

Synopsis

getfaceedges_returntype **GetRingEdges**(varchar atopology, integer aring, integer max_edges=null);

Descrição

Retorna o conjunto ordenado de identificadores de limites assinados, conhecidos caminhando em um lado da beirada. Cada saída consiste em uma sequência e uma id limite assinada. Números em sequência começam com o valor 1.

Se você passa uma id limite positiva, a caminhada começa no lado esquerdo do limite correspondente e segue sua direção. Se você passa uma id limite negativa, a caminhada começa no lado direito dele e orienta-se para trás.

Se `max_edges` não é nulo, não mais que aqueles relatos são retornados pela função. Isto foi feito para ser um parâmetro seguro ao lidar com topologias possivelmente inválidas.

Note
Esta função utiliza anel limite vinculando metadados.

Disponibilidade: 2.0.0

Veja também

ST_GetFaceEdges, GetNodeEdges

8.7.11 GetNodeEdges

GetNodeEdges — Retorna um conjunto ordenado de limites incidentes no dado nó.

Synopsis

getfaceedges_returntype **GetNodeEdges**(varchar atopology, integer anode);

Descrição

Retorna um conjunto de limites incidentes no dado nó. Cada saída consiste em uma sequência e uma id limite assinada. Os números sequência começam com o valor 1. Um limite positivo começa no dado nó. Um limite negativo termina no dado nó. Limites fechado aparecerão duas vezes (com ambos sinais). A ordem é sentido horário, começando do norte.

Note
Esta função computa ordenação em vez de derivação dos metadados e é, assim, útil para construir o vínculo do limite anel.

Disponibilidade: 2.0

Veja também

getfaceedges_returntype, GetRingEdges, ST_Azimuth

8.8 Processamento de Topologia

8.8.1 Polygonize

Polygonize — Finds and registers all faces defined by topology edges.

Synopsis

text **Polygonize**(varchar toponame);

Descrição

Registers all faces that can be built out a topology edge primitives.

A topologia alvo supostamente contém nenhuma borda que se auto intersecta.

Note
Faces já conhecidas são reconhecidas, logo, é seguro chamar Polygonize várias vezes na mesma topologia.

Note
Esta função não utiliza os campos set the next_left_edge e next_right_edge da table limite.

Disponibilidade: 2.0.0

Veja também

AddFace, ST_Polygonize

8.8.2 AddNode

AddNode — Adiciona um ponto nó na table nó no esquema topológico específico e retorna a nodeid do novo nó. Se o ponto já existe, a nodeid é retornada.

Synopsis

integer **AddNode**(varchar toponame, geometry apoint, boolean allowEdgeSplitting=false, boolean computeContainingFace=false);

Descrição

Adiciona um ponto nó na table nó no esquema topológico específico. A função AddEdge automaticamente adiciona pontos de início e fim de um limite quando chamado, não é necessário adicionar nós de um limite explicitamente.

Se qualquer limite cruzando o nó é encontrado, ou uma exceção surge ou a borda é dividida, dependendo do valor do parâmetro `allowEdgeSplitting`.

Se `computeContainingFace` for verdade, um novo nó adicionado irá corrigir a face computada.

Note
Se a geometria `apoint` já existe como um nó, não se adiciona um nó, mas a nodeid existente retorna.

Disponibilidade: 2.0.0

Exemplos

```
SELECT topology.AddNode('ma_topo', ST_GeomFromText('POINT(227641.6 893816.5)', 26986) ) As ↩
    nodeid;
-- result --
nodeid
--------
 4
```

Veja também

AddEdge, Cria topologia

8.8.3 AddEdge

AddEdge — Adiciona uma linestring limite à edge table e os pontos de início e fim associados à table ponto nó do esquema de topologia especificado usando a linestring geometria específica e retorna a bordaid da nova borda (ou da borda já existente).

Synopsis

integer **AddEdge**(varchar toponame, geometry aline);

Descrição

Adiciona uma borda à edge table e nós associados às nodes tables do esquema `toponame` especificado, usando a linestring geometria específica e retorna a bordaid do novo ou já existente relato. A nova borda adicionada tem a face "universal" nos dois lados e se conecta com si mesma.

Note
Se a `aline` geometria cruza, sobrepõe, contém ou é contida por uma borda linestring, um erro é lançado e a borda não é adicionada.

Note
A geometria da `aline` deve ter o mesmo `srid` definido para a topologia, senão um erro inválido é lançado no sistema de referência espacial.

Desempenhado pelo módulo GEOS.

Disponibilidade: 2.0.0

Exemplos

```
SELECT topology.AddEdge('ma_topo', ST_GeomFromText('LINESTRING(227575.8 893917.2,227591.9  ↩
    893900.4)', 26986) ) As edgeid;
-- result-
edgeid
--------
 1

SELECT topology.AddEdge('ma_topo', ST_GeomFromText('LINESTRING(227591.9 893900.4,227622.6  ↩
    893844.2,227641.6 893816.5,
 227704.5 893778.5)', 26986) ) As edgeid;
-- result --
edgeid
--------
 2

 SELECT topology.AddEdge('ma_topo', ST_GeomFromText('LINESTRING(227591.2 893900, 227591.9  ↩
     893900.4,
  227704.5 893778.5)', 26986) ) As edgeid;
 -- gives error --
 ERROR:  Edge intersects (not on endpoints) with existing edge 1
```

Veja também

TopoGeo_AddLineString, Cria topologia, Section 4.5

8.8.4 AddFace

AddFace — Registra uma face primitiva a uma topologia e pega seu identificador.

Synopsis

integer **AddFace**(varchar toponame, geometry apolygon, boolean force_new=false);

Descrição

Registra uma face primitiva a uma topologia e pega seu identificador.

Para uma nova face adicionada, as bordas formando seus limites e as contidas na face, serão atualizadas para ter valore corretos nos campos left_face e right_face. Os nós isolados contidos na face também serão atualizados para ter um valor correto do campo containing_face.

Note
Esta função não utiliza os campos set the next_left_edge e next_right_edge da table limite.

A topologia alvo é supostamente válida (não contendo nenhuma borda auto intersectada). Uma exceção surge se: O limite do polígono não estiver completamente definido ou caso o polígono sobreponha uma face existente.

Se a `apolygon` geometria já existe como face, então: se `force_new` é falso (o padrão) a id da face da face existente retorna, se `force_new` é verdade uma nova id será assinada para a nova face registrada.

Note
Quando um no registro de uma face existente é representada (force_new=true), nenhuma ação será tomada para resolver referências pendentes a face existente na borda, nó e tables relacionadas, nem o relato do campo MBR será atualizado. Fica a critério do chamador lidar ou não com isso.

Note
A geometria da `apolygon` deve ter o mesmo `srid` definido para a topologia, senão um erro inválido é lançado no sistema de referência espacial.

Disponibilidade: 2.0.0

Exemplos

```
-- first add the edges we use generate_series as an iterator (the below
-- will only work for polygons with < 10000 points because of our max in gs)
SELECT topology.AddEdge('ma_topo', ST_MakeLine(ST_PointN(geom,i), ST_PointN(geom, i + 1) )) ↩
     As edgeid
    FROM (SELECT  ST_NPoints(geom) AS npt, geom
            FROM
                (SELECT ST_Boundary(ST_GeomFromText('POLYGON((234896.5 899456.7,234914  ↩
                    899436.4,234946.6 899356.9,234872.5 899328.7,
                234891 899285.4,234992.5 899145, 234890.6 899069,234755.2 899255.4,
                234612.7 899379.4,234776.9 899563.7,234896.5 899456.7))', 26986) )  As geom
            )  As geoms) As facen CROSS JOIN generate_series(1,10000) As i
         WHERE i < npt;
-- result --
 edgeid
--------
      3
      4
      5
      6
      7
      8
      9
     10
     11
     12
(10 rows)
-- then add the face -

SELECT topology.AddFace('ma_topo',
    ST_GeomFromText('POLYGON((234896.5 899456.7,234914 899436.4,234946.6 899356.9,234872.5  ↩
        899328.7,
```

```
    234891 899285.4,234992.5 899145, 234890.6 899069,234755.2 899255.4,
    234612.7 899379.4,234776.9 899563.7,234896.5 899456.7))', 26986) ) As faceid;
-- result --
faceid
--------
 1
```

Veja também

AddEdge, Cria topologia, Section 4.5

8.8.5 ST_Simplify

ST_Simplify — Retorna uma versão "simplificada" da geometria da dada TopoGeometria usando o algorítimo Douglas-Peucker.

Synopsis

geometry **ST_Simplify**(TopoGeometry tg, float8 tolerance);

Descrição

Retorna uma versão "simplificada" da geometria da dada TopoGeometria usando o algorítimo Douglas-Peucker em cada borda componente.

Note
A geometria retornada pode ser não simples ou não válida.
Dividir bordas componentes pode ajudar a manter simplicidade/validade.

Desempenhado pelo módulo GEOS.

Disponibilidade: 2.1.0

Veja também

Geometry ST_Simplify, ST_IsSimple, ST_IsValid, ST_ModEdgeSplit

8.8.6 RemoveUnusedPrimitives

RemoveUnusedPrimitives — Removes topology primitives which not needed to define existing TopoGeometry objects.

Synopsis

int **RemoveUnusedPrimitives**(text topology_name, geometry bbox);

Descrição

Finds all primitives (nodes, edges, faces) that are not strictly needed to represent existing TopoGeometry objects and removes them, maintaining topology validity (edge linking, face labeling) and TopoGeometry space occupation.

No new primitive identifiers are created, but rather existing primitives are expanded to include merged faces (upon removing edges) or healed edges (upon removing nodes).

Availability: 3.3.0

Veja também

ST_ModEdgeHeal, ST_RemEdgeModFace

8.9 Construtores de TopoGeometria

8.9.1 CreateTopoGeom

CreateTopoGeom — Cria uma novo objeto de topo geometria de um arranjo topo elemento - tg_type: 1:[multi]point, 2:[multi]line, 3:[multi]poly, 4:collection

Synopsis

topogeometry **CreateTopoGeom**(varchar toponame, integer tg_type, integer layer_id, topoelementarray tg_objs);
topogeometry **CreateTopoGeom**(varchar toponame, integer tg_type, integer layer_id);

Descrição

Creates a topogeometry object for layer denoted by `layer_id` and registers it in the relations table in the `toponame` schema.

`tg_type` is an integer: 1:[multi]point (punctal), 2:[multi]line (lineal), 3:[multi]poly (areal), 4:collection. `layer_id` is the layer id in the topology.layer table.

camadas pontuais são formadas a partir de um conjunto de nós, camadas lineares são formadas a partir de um conjunto de bordas, camadas areais são formadas a partir de um conjunto de faces e as coleções podem ser formadas a partir de uma mistura de nós, bordas e faces.

Omitir o arranjo de componentes gera um objeto TopoGeometria vazio.

Availability: 1.1

Exemplos: Formados de bordas existentes

Create a topogeom in ri_topo schema for layer 2 (our ri_roads), of type (2) LINE, for the first edge (we loaded in `ST_CreateTopoGeo`)

```
INSERT INTO ri.ri_roads(road_name, topo) VALUES('Unknown', topology.CreateTopoGeom('ri_topo ↩
    ',2,2,'{{1,2}}'::topology.topoelementarray);
```

Exemplos: Converte uma geometria areal para uma topogeometria melhor

Digamos que tenhamos geometrias que deveriam ser formadas de uma coleção de faces. Nós temos, por exemplo, blockgroupos tables e queremos saber a topo geometria de cada block group. Se seus dados foram perfeitamente alinhados, podemos fazer isto:

```
-- create our topo geometry column --
SELECT topology.AddTopoGeometryColumn(
        'topo_boston',
        'boston', 'blockgroups', 'topo', 'POLYGON');

-- addtopgeometrycolumn --
1

-- update our column assuming
-- everything is perfectly aligned with our edges
UPDATE boston.blockgroups AS bg
        SET topo = topology.CreateTopoGeom('topo_boston'
        ,3,1
```

```
        , foo.bfaces)
FROM (SELECT b.gid,  topology.TopoElementArray_Agg(ARRAY[f.face_id,3]) As bfaces
        FROM boston.blockgroups As b
            INNER JOIN topo_boston.face As f ON b.geom && f.mbr
        WHERE ST_Covers(b.geom, topology.ST_GetFaceGeometry('topo_boston', f.face_id))
            GROUP BY b.gid) As foo
WHERE foo.gid = bg.gid;
```

```
--the world is rarely perfect allow for some error
--count the face if 50% of it falls
-- within what we think is our blockgroup boundary
UPDATE boston.blockgroups AS bg
        SET topo = topology.CreateTopoGeom('topo_boston'
        ,3,1
        , foo.bfaces)
FROM (SELECT b.gid,  topology.TopoElementArray_Agg(ARRAY[f.face_id,3]) As bfaces
        FROM boston.blockgroups As b
            INNER JOIN topo_boston.face As f ON b.geom && f.mbr
        WHERE ST_Covers(b.geom, topology.ST_GetFaceGeometry('topo_boston', f.face_id))
        OR
 (  ST_Intersects(b.geom, topology.ST_GetFaceGeometry('topo_boston', f.face_id))
            AND ST_Area(ST_Intersection(b.geom, topology.ST_GetFaceGeometry('topo_boston',  ↩
                f.face_id) ) ) >
                ST_Area(topology.ST_GetFaceGeometry('topo_boston', f.face_id))*0.5
                )
            GROUP BY b.gid) As foo
WHERE foo.gid = bg.gid;

-- and if we wanted to convert our topogeometry back
-- to a denormalized geometry aligned with our faces and edges
-- cast the topo to a geometry
-- The really cool thing is my new geometries
-- are now aligned with my tiger street centerlines
UPDATE boston.blockgroups SET new_geom = topo::geometry;
```

Veja também

AddTopoGeometryColumn, toTopoGeom ST_CreateTopoGeo, ST_GetFaceGeometry, TopoElementArray, TopoElementArray_Agg

8.9.2 toTopoGeom

toTopoGeom — Converts a simple Geometry into a topo geometry.

Synopsis

topogeometry **toTopoGeom**(geometry geom, varchar toponame, integer layer_id, float8 tolerance);
topogeometry **toTopoGeom**(geometry geom, topogeometry topogeom, float8 tolerance);

Descrição

Converte uma simples geometria em TopoGeometry.

Topológicos primitivos requeridos para representar a geometria de entrada será adicionada a topologia oculta, possivelmente dividindo as existentes, e elas serão associadas com a TopoGeometria de saída na table `relation`.

Objetos existentes de TopoGeometria (com a possível exceção de `topogeom`, se dada) manterão suas formas.

Quando `tolerance` é dada, será usada para quebrar a geometria de entrada para primitivas existentes.

Na primeira forma, uma nova TopoGeometria será criada para a dada camada (`layer_id`) da topologia (`toponame`)

Na segunda forma, as rpimitivs resultantes da conversão serão adicionadas a uma TopoGeometria pre existente (`topogeom`), adicionando, possivelmente, espaço à sua forma final. Para obter a nova forma completamente substituir a antiga, veja clearTopoGeom.

Disponibilidade: 2.0

Melhorias: 2.1.0 adiciona a versão pegando uma TopoGeometria existente.

Exemplos

Este é um fluxo de trabalho auto contido completo

```
-- do this if you don't have a topology setup already
-- creates topology not allowing any tolerance
SELECT topology.CreateTopology('topo_boston_test', 2249);
-- create a new table
CREATE TABLE nei_topo(gid serial primary key, nei varchar(30));
--add a topogeometry column to it
SELECT topology.AddTopoGeometryColumn('topo_boston_test', 'public', 'nei_topo', 'topo', ' ↩
    MULTIPOLYGON') As new_layer_id;
new_layer_id
------------
1

--use new layer id in populating the new topogeometry column
-- we add the topogeoms to the new layer with 0 tolerance
INSERT INTO nei_topo(nei, topo)
SELECT nei,  topology.toTopoGeom(geom, 'topo_boston_test', 1)
FROM neighborhoods
WHERE gid BETWEEN 1 and 15;

--use to verify what has happened --
SELECT * FROM
    topology.TopologySummary('topo_boston_test');

-- summary--
Topology topo_boston_test (5), SRID 2249, precision 0
61 nodes, 87 edges, 35 faces, 15 topogeoms in 1 layers
Layer 1, type Polygonal (3), 15 topogeoms
 Deploy: public.nei_topo.topo
```

```
-- Shrink all TopoGeometry polygons by 10 meters
UPDATE nei_topo SET topo = ST_Buffer(clearTopoGeom(topo), -10);

-- Get the no-one-lands left by the above operation
-- I think GRASS calls this "polygon0 layer"
SELECT ST_GetFaceGeometry('topo_boston_test', f.face_id)
  FROM topo_boston_test.face f
  WHERE f.face_id
> 0 -- don't consider the universe face
  AND NOT EXISTS ( -- check that no TopoGeometry references the face
    SELECT * FROM topo_boston_test.relation
    WHERE layer_id = 1 AND element_id = f.face_id
  );
```

Veja também

Cria topologia, AddTopoGeometryColumn, CreateTopoGeom, TopologySummary, clearTopoGeom

8.9.3 TopoElementArray_Agg

TopoElementArray_Agg — Returns a `topoelementarray` for a set of element_id, type arrays (topoelements).

Synopsis

topoelementarray **TopoElementArray_Agg**(topoelement set tefield);

Descrição

Usado para criar um TopoElementArray de um conjunto de TopoElement.

Disponibilidade: 2.0.0

Exemplos

```
SELECT topology.TopoElementArray_Agg(ARRAY[e,t]) As tea
  FROM generate_series(1,3) As e CROSS JOIN generate_series(1,4) As t;
  tea
---------------------------------------------------------------------------
{{1,1},{1,2},{1,3},{1,4},{2,1},{2,2},{2,3},{2,4},{3,1},{3,2},{3,3},{3,4}}
```

Veja também

TopoElement, TopoElementArray

8.9.4 TopoElement

TopoElement — Converts a topogeometry to a topoelement.

Synopsis

topoelement **TopoElement**(topogeometry topo);

Descrição

Converts a TopoGeometry to a TopoElement.

Availability: 3.4.0

Exemplos

Este é um fluxo de trabalho auto contido completo

```
-- do this if you don't have a topology setup already
-- Creates topology not allowing any tolerance
SELECT TopoElement(topo)
FROM neighborhoods;
```

```
-- using as cast
SELECT topology.TopoElementArray_Agg(topo::topoelement)
FROM neighborhoods
GROUP BY city;
```

Veja também

TopoElementArray_Agg, TopoGeometry, TopoElement

8.10 Editores de TopoGeometria

8.10.1 clearTopoGeom

clearTopoGeom — Clears the content of a topo geometry.

Synopsis

topogeometry **clearTopoGeom**(topogeometry topogeom);

Descrição

Limpa o conteúdo de um TopoGeometry tornando-o vazio. Mais útil quando usado em conjunto com toTopoGeom para substituir o formato de objetos existentes e qualquer objeto dependente em níveis hierárquicos mais elevados.

Disponibilidade: 2.1

Exemplos

```
-- Shrink all TopoGeometry polygons by 10 meters
UPDATE nei_topo SET topo = ST_Buffer(clearTopoGeom(topo), -10);
```

Veja também

toTopoGeom

8.10.2 TopoGeom_addElement

TopoGeom_addElement — Adds an element to the definition of a TopoGeometry.

Synopsis

topogeometry **TopoGeom_addElement**(topogeometry tg, topoelement el);

Descrição

Adiciona um TopoElement à definição de um objeto de TopoGeometria. Não apresenta erro se o elemento já faz parte da definição.

Disponibilidade: 2.3

Exemplos

```
-- Add edge 5 to TopoGeometry tg
UPDATE mylayer SET tg = TopoGeom_addElement(tg, '{5,2}');
```

Veja também

TopoGeom_remElement, CreateTopoGeom

8.10.3 TopoGeom_remElement

TopoGeom_remElement — Removes an element from the definition of a TopoGeometry.

Synopsis

topogeometry **TopoGeom_remElement**(topogeometry tg, topoelement el);

Descrição

Remove um TopoElement de uma definição de uma objeto de TopoGeometrias.

Disponibilidade: 2.3

Exemplos

```
-- Remove face 43 from TopoGeometry tg
UPDATE mylayer SET tg = TopoGeom_remElement(tg, '{43,3}');
```

Veja também

TopoGeom_addElement, CreateTopoGeom

8.10.4 TopoGeom_addTopoGeom

TopoGeom_addTopoGeom — Adds element of a TopoGeometry to the definition of another TopoGeometry.

Synopsis

topogeometry **TopoGeom_addTopoGeom**(topogeometry tgt, topogeometry src);

Descrição

Adds the elements of a TopoGeometry to the definition of another TopoGeometry, possibly changing its cached type (type attribute) to a collection, if needed to hold all elements in the source object.

The two TopoGeometry objects need be defined against the *same* topology and, if hierarchically defined, need be composed by elements of the same child layer.

Availability: 3.2

Exemplos

```
-- Set an "overall" TopoGeometry value to be composed by all
-- elements of specific TopoGeometry values
UPDATE mylayer SET tg_overall = TopoGeom_addTopogeom(
    TopoGeom_addTopoGeom(
        clearTopoGeom(tg_overall),
        tg_specific1
    ),
    tg_specific2
);
```

Veja também

TopoGeom_addElement, clearTopoGeom, CreateTopoGeom

8.10.5 toTopoGeom

toTopoGeom — Adds a geometry shape to an existing topo geometry.

Descrição

Refer to toTopoGeom.

8.11 Assessores de TopoGeometria

8.11.1 GetTopoGeomElementArray

GetTopoGeomElementArray — Returns a `topoelementarray` (an array of topoelements) containing the topological elements and type of the given TopoGeometry (primitive elements).

Synopsis

topoelementarray **GetTopoGeomElementArray**(varchar toponame, integer layer_id, integer tg_id);

topoelementarray **GetTopoGeomElementArray**(topogeometry tg);

Descrição

Retorna um TopoElementArray contendo os tipos e elementos da dada TopoGeometria (elementos primitivos). Isto é parecido com GetTopoGeomElements, mas em vez de retornar os elementos como dataset, retorna como um arranjo.

tg_id é a id topogeometria do objeto de topogeometria na camada indicada pela `layer_id` na topology.layer table.

Availability: 1.1

Exemplos

Veja também

GetTopoGeomElements, TopoElementArray

8.11.2 GetTopoGeomElements

GetTopoGeomElements — Returns a set of `topoelement` objects containing the topological element_id,element_type of the given TopoGeometry (primitive elements).

Synopsis

setof topoelement **GetTopoGeomElements**(varchar toponame, integer layer_id, integer tg_id);

setof topoelement **GetTopoGeomElements**(topogeometry tg);

Descrição

Returns a set of element_id,element_type (topoelements) corresponding to primitive topology elements TopoElement (1: nodes, 2: edges, 3: faces) that a given topogeometry object in `toponame` schema is composed of.

tg_id é a id topogeometria do objeto de topogeometria na camada indicada pela `layer_id` na topology.layer table.

Disponibilidade: 2.0.0

Exemplos

Veja também

GetTopoGeomElementArray, TopoElement, TopoGeom_addElement, TopoGeom_remElement

8.11.3 ST_SRID

ST_SRID — Returns the spatial reference identifier for a topogeometry.

Synopsis

integer **ST_SRID**(topogeometry tg);

Descrição

Returns the spatial reference identifier for the ST_Geometry as defined in spatial_ref_sys table. Section 4.5

Note
spatial_ref_sys table is a table that catalogs all spatial reference systems known to PostGIS and is used for transformations from one spatial reference system to another. So verifying you have the right spatial reference system identifier is important if you plan to ever transform your geometries.

Availability: 3.2.0


This method implements the SQL/MM specification. SQL-MM 3: 14.1.5

Exemplos

```
SELECT ST_SRID(ST_GeomFromText('POINT(-71.1043 42.315)',4326));
                --result
                4326
```

Veja também

Section 4.5, ST_SetSRID, ST_Transform, ST_SRID

8.12 TopoGeometry Outputs

8.12.1 AsGML

AsGML — Retorna a representação GML de uma topogeometria.

Synopsis

text **AsGML**(topogeometry tg);
text **AsGML**(topogeometry tg, text nsprefix_in);
text **AsGML**(topogeometry tg, regclass visitedTable);
text **AsGML**(topogeometry tg, regclass visitedTable, text nsprefix);
text **AsGML**(topogeometry tg, text nsprefix_in, integer precision, integer options);
text **AsGML**(topogeometry tg, text nsprefix_in, integer precision, integer options, regclass visitedTable);
text **AsGML**(topogeometry tg, text nsprefix_in, integer precision, integer options, regclass visitedTable, text idprefix);
text **AsGML**(topogeometry tg, text nsprefix_in, integer precision, integer options, regclass visitedTable, text idprefix, int gmlversion);

Descrição

Retorna a representação GML de uma topogeometria na versão GML3 format. Se o `nsprefix_in` não for especificado, então `gml` é usado. Passa em uma string vazia para nsprefix para pegar um espaço não qualificado. A precisão (padrão: 15) e parâmetros (padrão 1) de opções, se dados, são passados inalterados para a chamada subjacente para ST_AsGML.

O parâmetro `visitedTable`, se dado, é usado para manter o caminho dos elementos nó e borda visitados, assim como para usar referências-cruzadas (xlink:xref) em vez de definições duplicadas. É esperado que a tabela tenha (pelo menos) dois campos inteiros: 'element_type' e 'element_id'. O usuário visitante deve ter os privilégios escritos e lidos na dada tabela. Para uma melhor apresentação, um index deve ser definido no `element_type` e `element_id`, nesta ordem. Tal index será adicionado automaticamente ao adicionar uma única limitação aos campos. Exemplo:

```
CREATE TABLE visited (
  element_type integer, element_id integer,
  unique(element_type, element_id)
);
```

O parâmetro `idprefix`, se dado, será antecipado aos identificadores tag de bordas e nós.

O parâmetro `gmlver`, se dado, será passado à ST_AsGML subjacente. Padrões para 3.

Disponibilidade: 2.0.0

Exemplos

Isto usa a topo geometria que criamos em CreateTopoGeom

```
SELECT topology.AsGML(topo) As rdgml
  FROM ri.roads
  WHERE road_name = 'Unknown';

-- rdgml--
<gml:TopoCurve>
    <gml:directedEdge>
```

```
        <gml:Edge gml:id="E1">
            <gml:directedNode orientation="-">
                <gml:Node gml:id="N1"/>
            </gml:directedNode>
            <gml:directedNode
></gml:directedNode>
            <gml:curveProperty>
                <gml:Curve srsName="urn:ogc:def:crs:EPSG::3438">
                    <gml:segments>
                        <gml:LineStringSegment>
                            <gml:posList srsDimension="2"
>384744 236928 384750 236923 384769 236911 384799 236895 384811 236890
                            384833 236884 384844 236882 384866 236881 384879 236883 384954 ↩
                                236898 385087 236932 385117 236938
                            385167 236938 385203 236941 385224 236946 385233 236950 385241 ↩
                                236956 385254 236971
                            385260 236979 385268 236999 385273 237018 385273 237037 385271 ↩
                                237047 385267 237057 385225 237125
                            385210 237144 385192 237161 385167 237192 385162 237202 385159 ↩
                                237214 385159 237227 385162 237241
                            385166 237256 385196 237324 385209 237345 385234 237375 385237 ↩
                                237383 385238 237399 385236 237407
                            385227 237419 385213 237430 385193 237439 385174 237451 385170 ↩
                                237455 385169 237460 385171 237475
                            385181 237503 385190 237521 385200 237533 385206 237538 385213 ↩
                                237541 385221 237542 385235 237540 385242 237541
                            385249 237544 385260 237555 385270 237570 385289 237584 385292 ↩
                                237589 385291 237596 385284 237630</gml:posList>
                        </gml:LineStringSegment>
                    </gml:segments>
                </gml:Curve>
            </gml:curveProperty>
        </gml:Edge>
    </gml:directedEdge>
</gml:TopoCurve
>
```

É o mesmo exercício do o anterior, mas sem o espaço para nome

```
SELECT topology.AsGML(topo,'') As rdgml
  FROM ri.roads
  WHERE road_name = 'Unknown';

-- rdgml--
<TopoCurve>
    <directedEdge>
        <Edge id="E1">
            <directedNode orientation="-">
                <Node id="N1"/>
            </directedNode>
            <directedNode
></directedNode>
            <curveProperty>
                <Curve srsName="urn:ogc:def:crs:EPSG::3438">
                    <segments>
                        <LineStringSegment>
                            <posList srsDimension="2"
>384744 236928 384750 236923 384769 236911 384799 236895 384811 236890
                            384833 236884 384844 236882 384866 236881 384879 236883 384954 ↩
                                236898 385087 236932 385117 236938
                            385167 236938 385203 236941 385224 236946 385233 236950 385241 ↩
                                236956 385254 236971
```

```
                            385260 236979 385268 236999 385273 237018 385273 237037 385271  ↩
                                237047 385267 237057 385225 237125
                            385210 237144 385192 237161 385167 237192 385162 237202 385159  ↩
                                237214 385159 237227 385162 237241
                            385166 237256 385196 237324 385209 237345 385234 237375 385237  ↩
                                237383 385238 237399 385236 237407
                            385227 237419 385213 237430 385193 237439 385174 237451 385170  ↩
                                237455 385169 237460 385171 237475
                            385181 237503 385190 237521 385200 237533 385206 237538 385213  ↩
                                237541 385221 237542 385235 237540 385242 237541
                            385249 237544 385260 237555 385270 237570 385289 237584 385292  ↩
                                237589 385291 237596 385284 237630</posList>
                        </LineStringSegment>
                    </segments>
                </Curve>
            </curveProperty>
        </Edge>
    </directedEdge>
</TopoCurve
>
```

Veja também

CreateTopoGeom, ST_CreateTopoGeo

8.12.2 AsTopoJSON

AsTopoJSON — Retorna a representação TopoJSON de uma topogeometria.

Synopsis

text **AsTopoJSON**(topogeometry tg, regclass edgeMapTable);

Descrição

Retorna a representação TopoJSON de uma topogeometria. Se a `edgeMapTable` não for nula, será usada como um mapeamento de pesquisa/armazenamento de identificadores de borda para índices de arcos. Isto é para permitir um arranjo compacto de "arcos" no documento final.

É esperado que a tabela, se dada, tenha um campo "arc_id" do tipo "serial" e uma "edge_id" de tipo inteiro; o código irá consultar a tabela para "edge_id", então é recomendado adicionar um index naquele campo.

Note
Os índices de arcos na saída TopoJSON são 0-baseados mas são 1-baseados na tabela "edgeMapTable".

Um documento TopoJSON completo precisará conter, em soma com os fragmentos retornados por esta função, os arcos atuais mais alguns cabeçalhos. Veja a TopoJSON specification.

Disponibilidade: 2.1.0

Melhorias: 2.2.1 suporte para entradas pontuais adicionado

Veja também

ST_AsGeoJSON

Exemplos

```
CREATE TEMP TABLE edgemap(arc_id serial, edge_id int unique);

-- header
SELECT '{ "type": "Topology", "transform": { "scale": [1,1], "translate": [0,0] }, "objects ↩
    ": {'

-- objects
UNION ALL SELECT '"' || feature_name || '": ' || AsTopoJSON(feature, 'edgemap')
FROM features.big_parcels WHERE feature_name = 'P3P4';

-- arcs
WITH edges AS (
  SELECT m.arc_id, e.geom FROM edgemap m, city_data.edge e
  WHERE e.edge_id = m.edge_id
), points AS (
  SELECT arc_id, (st_dumppoints(geom)).* FROM edges
), compare AS (
  SELECT p2.arc_id,
         CASE WHEN p1.path IS NULL THEN p2.geom
              ELSE ST_Translate(p2.geom, -ST_X(p1.geom), -ST_Y(p1.geom))
         END AS geom
  FROM points p2 LEFT OUTER JOIN points p1
  ON ( p1.arc_id = p2.arc_id AND p2.path[1] = p1.path[1]+1 )
  ORDER BY arc_id, p2.path
), arcsdump AS (
  SELECT arc_id, (regexp_matches( ST_AsGeoJSON(geom), '\[.*\]'))[1] as t
  FROM compare
), arcs AS (
  SELECT arc_id, '[' || array_to_string(array_agg(t), ',') || ']' as a FROM arcsdump
  GROUP BY arc_id
  ORDER BY arc_id
)
SELECT '}, "arcs": [' UNION ALL
SELECT array_to_string(array_agg(a), E',\n') from arcs

-- footer
UNION ALL SELECT ']}'::text as t;

-- Result:
{ "type": "Topology", "transform": { "scale": [1,1], "translate": [0,0] }, "objects": {
"P3P4": { "type": "MultiPolygon", "arcs": [[[-1]],[[6,5,-5,-4,-3,1]]]}
}, "arcs": [
 [[25,30],[6,0],[0,10],[-14,0],[0,-10],[8,0]],
 [[35,6],[0,8]],
 [[35,6],[12,0]],
 [[47,6],[0,8]],
 [[47,14],[0,8]],
 [[35,22],[12,0]],
 [[35,14],[0,8]]
 ]}
```

8.13 Relações de Topologia Espacial

8.13.1 Equivalentes

Equivalentes — Retorna verdade se duas topogeometrias forem compostas da mesma topologia primitiva

Synopsis

boolean **Equals**(topogeometry tg1, topogeometry tg2);

Descrição

Retorna verdade se duas topogeometrias forem compostas das mesmas topologias primitivas: faces, bordas e nós.

Note
Esta função não é suportada por geometrias que são coleções de geometrias. Também não pode comparar topogeometrias de topologias diferentes.

Disponibilidade: 1.1.0


This function supports 3d and will not drop the z-index.

Exemplos

Veja também

GetTopoGeomElements, ST_Equals

8.13.2 Intercepta

Intercepta — Retorna verdade se algum par de primitivos das duas topologias se intersectar.

Synopsis

boolean **Intersects**(topogeometry tg1, topogeometry tg2);

Descrição

Retorna verdade se algum par de primitivos das duas topologias se intersectar.

Note
This function not supported for topogeometries that are geometry collections. It also can not compare topogeometries from different topologies. Also not currently supported for hierarchical topogeometries (topogeometries composed of other topogeometries).

Disponibilidade: 1.1.0


This function supports 3d and will not drop the z-index.

Exemplos

Veja também

ST_Intersects

8.14 Importing and exporting Topologies

Once you have created topologies, and maybe associated topological layers, you might want to export them into a file-based format for backup or transfer into another database.

Using the standard dump/restore tools of PostgreSQL is problematic because topologies are composed by a set of tables (4 for primitives, an arbitrary number for layers) and records in metadata tables (topology.topology and topology.layer). Additionally, topology identifiers are not univoque across databases so that parameter of your topology will need to be changes upon restoring it.

In order to simplify export/restore of topologies a pair of executables are provided: `pgtopo_export` and `pgtopo_import`. Example usage:

```
pgtopo_export dev_db topo1 | pgtopo_import topo1 | psql staging_db
```

8.14.1 Using the Topology exporter

The `pgtopo_export` script takes the name of a database and a topology and outputs a dump file which can be used to import the topology (and associated layers) into a new database.

By default `pgtopo_export` writes the dump file to the standard output so that it can be piped to `pgtopo_import` or redirected to a file (refusing to write to terminal). You can optionally specify an output filename with the `-f` commandline switch.

By default `pgtopo_export` includes a dump of all layers defined against the given topology. This may be more data than you need, or may be non-working (in case your layer tables have complex dependencies) in which case you can request skipping the layers with the `--skip-layers` switch and deal with those separately.

Invoking `pgtopo_export` with the `--help` (or `-h` for short) switch will always print short usage string.

The dump file format is a compressed tar archive of a `pgtopo_export` directory containing at least a `pgtopo_dump_version` file with format version info. As of version `1` the directory contains tab-delimited CSV files with data of the topology primitive tables (node, edge_data, face, relation), the topology and layer records associated with it and (unless `--skip-layers` is given) a custom-format PostgreSQL dump of tables reported as being layers of the given topology.

8.14.2 Using the Topology importer

The `pgtopo_import` script takes a `pgtopo_export` format topology dump and a name to give to the topology to be created and outputs an SQL script reconstructing the topology and associated layers.

The generated SQL file will contain statements that create a topology with the given name, load primitive data in it, restores and registers all topology layers by properly linking all TopoGeometry values to their correct topology.

By default `pgtopo_import` reads the dump from the standard input so that it can be used in conjuction with `pgtopo_export` in a pipeline. You can optionally specify an input filename with the `-f` commandline switch.

By default `pgtopo_import` includes in the output SQL file the code to restore all layers found in the dump.

This may be unwanted or non-working in case your target database already have tables with the same name as the ones in the dump. In that case you can request skipping the layers with the `--skip-layers` switch and deal with those separately (or later).

SQL to only load and link layers to a named topology can be generated using the `--only-layers` switch. This can be useful to load layers AFTER resolving the naming conflicts or to link layers to a different topology (say a spatially-simplified version of the starting topology).

Chapter 9

Gerência de dados raster, pesquisas e aplicações

9.1 Carregando e criando dados matriciais

Para a maioria dos casos, você usará a ferramenta `raster2pgsql` para carregar os dados matricias para o PostGIS.

9.1.1 Usando o raster2pgsql para carregar dados matricias

The `raster2pgsql` is a raster loader executable that loads GDAL supported raster formats into SQL suitable for loading into a PostGIS raster table. It is capable of loading folders of raster files as well as creating overviews of rasters.

Since the raster2pgsql is compiled as part of PostGIS most often (unless you compile your own GDAL library), the raster types supported by the executable will be the same as those compiled in the GDAL dependency library. To get a list of raster types your particular `raster2pgsql` supports use the `-G` switch.

Note
Na criação de overviews de um fator específico de um conjunto de rasters que estão alinhados, é possível que as overviews não se alinhem. Visite http://trac.osgeo.org/postgis/ticket/1764 para ver um exemplo onde as overviews não se alinham.

9.1.1.1 Example Usage

Uma sessão de exemplo usando o carregador para criar um arquivo de entrada fazendo upload de seus azulejos fragmentados em 100x100, pode ficar parecido com:

```
# -s use srid 4326
# -I create spatial index
# -C use standard raster constraints
# -M vacuum analyze after load
# *.tif load all these files
# -F include a filename column in the raster table
# -t tile the output 100x100
# public.demelevation load into this table
raster2pgsql -s 4326 -I -C -M -F -t 100x100 *.tif public.demelevation
> elev.sql

# -d connect to this database
# -f read this file after connecting
psql -d gisdb -f elev.sql
```

Note
If you do not specify the schema as part of the target table name, the table will be created in the default schema of the database or user you are connecting with.

Uma conversão e u upload podem ser feitos em apenas um passo usando encadeamento UNIX:

```
raster2pgsql -s 4326 -I -C -M *.tif -F -t 100x100 public.demelevation | psql -d gisdb
```

Carregue as tiles em metros dos rasters do estado plano de Massachusetts no esquema chamado: `aerial` e crie uma view completa,tabelas panoramas de níveis 2 e 4, use o modo cópia para inserir (sem arquivos intermediários), e -e não força tudo em uma transação (é bom se você quiser ver dados em tabelas sem esperar nada por isso). Quebre os rasters em 128x128 pixel tiles e aplique restrições de rasters. Utilize o modo cópia em vez da tabela inserida. (-F) Inclui um campo chamado filename para conter o nome do arquivo de onde as tiles cortam.

```
raster2pgsql -I -C -e -Y -F -s 26986 -t 128x128  -l 2,4 bostonaerials2008/*.jpg aerials. ↩
    boston | psql -U postgres -d gisdb -h localhost -p 5432
```

```
--get a list of raster types supported:
raster2pgsql -G
```

Os comandos -G geram uma lista parecida com

```
Available GDAL raster formats:
  Virtual Raster
  GeoTIFF
  National Imagery Transmission Format
  Raster Product Format TOC format
  ECRG TOC format
  Erdas Imagine Images (.img)
  CEOS SAR Image
  CEOS Image
  ...
  Arc/Info Export E00 GRID
  ZMap Plus Grid
  NOAA NGS Geoid Height Grids
```

9.1.1.2 raster2pgsql options

-? Tela de ajuda. A ajuda também é exibida se você não passar em nenhum argumento.

-G Imprimir os formatos de raster suportados.

(c|a|d|p) Essas são opções mutualmente exclusivas:

-c Criar nova table e popular ela com raster(s), *esse é o modo padrão*

-a Anexar raster(s) à uma table existente.

-d Derrubar table, criar nova e popular ela com raster(s)

-p Preparar modo, somente criar a table.

Processo raster: Solicitando restrições para registro apropriado nos catálogos raster

-C Solicitar restrições raster -- srid, pixelsize etc. para assegurar que o raster está registrado corretamente na view `raster_columns`.

-x Desativar configuração da extensão de restrição máxima. Empregado somente se a bandeira -C também for.

-r Configurar as restrições (espacialmente únicas e telha de cobertura) para bloqueio normal. Empregado somente se a bandeira -C também for.

Processo raster: Parâmetros opcionais usados para manipular a entrada do conjunto de dados raster

-s <SRID> Saída raster designada com SRID específico. Se não for fornecida ou for zero, os metadados raster serão verificados para determinar um SRID apropriado.

-b BAND Índice (1-base) da banda para extrair de raster. Para mais de um índice de banda, separe com vírgula(,). Se não especificado, todas as bandas de raster serão extraídas.

-t TILE_SIZE Corte raster em ladrilhos para ser inserido um por fileira na tabela. O `TILE_SIZE` é expressado como LARGURAxALTURA ou configurado para o valor "auto" para permitir o carregador computar um tamanho usando o primeiro raster e aplicando para os outros rasters.

-P Preenche a maioria das tiles da direita e de baixo para garantir que todas as tiles tenham a mesma largura e peso.

-R, --register Registrar o raster como o sistema de arquivos raster (out-db).

Somente os metadados do raster e a localização do caminho para o raster estão armazenados no banco de dados (não os pixels).

-l OVERVIEW_FACTOR Cria uma visão geral do raster. Para mais de um fator, separe com a vírgula(,). A visão geral da tabela de nomes segue o modelo o`_overview factor_table`, onde `overview factor` é um marcador de posição para um fator de visão geral numérico e `table` é substituído com a tabela de nome básica. A visão geral criada está armazenada no banco de dados e não é afetada por -R. Note que seu arquivo sql criado contém a tabela principal e as tabelas panoramas.

-N NODATA NODATA valor para usar em bandas sem um valor NODATA.

Parâmetros opcionais usados para manipular objetos do banco de dados

-f COLUMN Especificar nome de destinação da coluna raster, o padrão é 'rast'

-F Adicionar uma coluna com o nome do arquivo

-n COLUMN Especificar o nome da coluna filename. Sugere -F.

-q Identificadores wrap PostgreSQL em citações.

-I Cria um índice GiST na coluna raster.

-M Vácuo analise a tabela raster.

-k Keeps empty tiles and skips NODATA value checks for each raster band. Note you save time in checking, but could end up with far more junk rows in your database and those junk rows are not marked as empty tiles.

-T tablespace Especificar o espaço da tabela para a nova tabela. Note que os índices (incluindo a chave primária) continuarão sendo usados o espaço de tabela padrão, a menos que a bandeira -X também esteja sendo usada.

-X tablespace Especificar o espaço da tabela para a nova tabela. Isto se aplica à chave primária e ao índice espacial se a bandeira -I estiver sendo usada.

-Y max_rows_per_copy=50 Use copy statements instead of insert statements. Optionally specify `max_rows_per_copy`; default 50 when not specified.

-e Execute cada declaração individualmente, não use uma transação.

-E ENDIAN Controla endianidade da saída binária gerada do raster; especificamos 0 para XDR e 1 para NDR (padrão); somente a saída NDR é suportada agora

-V version Especificamos uma versão de formato de saída. O padrão é 0. Somente o 0 é suportado neste momento.

9.1.2 Criando rasters utilizando as funções rasters do PostGIS

Em várias ocasiões, você vai querer criar rasters e tabelas rasters no banco de dados. Existe uma superabundância de funções que fazem isto. Os passos gerais para seguir.

1. Cria uma tabela com uma coluna raster para segurar os novos relatos rasters que são efetuados com:

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

2. Existem várias funções para auxiliar com este objetivo. Se você não estiver criando rasters como derivados de outros rasters, você precisará de começar com: ST_MakeEmptyRaster, seguido por ST_AddBand

   Você também pode criar rasters de geometrias. Para alcançar seu objetivo, você irá querer usar ST_AsRaster talvez acompanhado com outras funções como: ST_Union ou ST_MapAlgebraFct ou qualquer um da família de outras funções álgebra de mapa.

   Existem ainda mais opções para a criação de novas tabelas rasters a partir das tabelas existentes. Você pode criar uma tabela raster em uma projeção diferente de uma existente, por exemplo, utilizando: ST_Transform

3. Uma vez que você houver terminado de popular sua tabela inicialmente, você vai querer criar um índice espacial na coluna raster com algo parecido com:

   ```
   CREATE INDEX myrasters_rast_st_convexhull_idx ON myrasters USING gist( ST_ConvexHull( ↩
       rast) );
   ```

   Note o uso do ST_ConvexHull já que a maioria dos operados raster são baseados no casco convexo dos rasters.

   Note
   Versões pre-2.0 do raster PostGIS eram baseadas no envelope em vez do casco convexo, Para os índices espaciais funcionar propriamente, você precisará derrubar estes e substituí-los com índice de casco convexo.

4. Aplique restrições rasters usando

9.1.3 Using "out db" cloud rasters

The `raster2pgsql` tool uses GDAL to access raster data, and can take advantage of a key GDAL feature: the ability to read from rasters that are stored remotely in cloud "object stores" (e.g. AWS S3, Google Cloud Storage).

Efficient use of cloud stored rasters requires the use of a "cloud optimized" format. The most well-known and widely used is the "cloud optimized GeoTIFF" format. Using a non-cloud format, like a JPEG, or an un-tiled TIFF will result in very poor performance, as the system will have to download the entire raster each time it needs to access a subset.

First, load your raster into the cloud storage of your choice. Once it is loaded, you will have a URI to access it with, either an "http" URI, or sometimes a URI specific to the service. (e.g., "s3://bucket/object"). To access non-public buckets, you will need to supply GDAL config options to authenticate your connection. Note that this command is *reading* from the cloud raster and *writing* to the database.

```
AWS_ACCESS_KEY_ID=xxxxxxxxxxxxxxxxxx \
AWS_SECRET_ACCESS_KEY=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx \
raster2pgsql \
  -s 990000 \
  -t 256x256 \
  -I \
  -R \
  /vsis3/your.bucket.com/your_file.tif \
  your_table \
  | psql your_db
```

Once the table is loaded, you need to give the database permission to read from remote rasters, by setting two permissions, postgis.enable_outdb_rasters and postgis.gdal_enabled_drivers.

```
SET postgis.enable_outdb_rasters = true;
SET postgis.gdal_enabled_drivers TO 'ENABLE_ALL';
```

To make the changes sticky, set them directly on your database. You will need to re-connect to experience the new settings.

```
ALTER DATABASE your_db SET postgis.enable_outdb_rasters = true;
ALTER DATABASE your_db SET postgis.gdal_enabled_drivers TO 'ENABLE_ALL';
```

For non-public rasters, you may have to provide access keys to read from the cloud rasters. The same keys you used to write the `raster2pgsql` call can be set for use inside the database, with the postgis.gdal_datapath configuration. Note that multiple options can be set by space-separating the `key=value` pairs.

```
SET postgis.gdal_vsi_options = 'AWS_ACCESS_KEY_ID=xxxxxxxxxxxxxxxxxxx
AWS_SECRET_ACCESS_KEY=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx';
```

Once you have the data loaded and permissions set you can interact with the raster table like any other raster table, using the same functions. The database will handle all the mechanics of connecting to the cloud data when it needs to read pixel data.

9.2 Catálogos Raster

Existem duas view raster catalogadas que vêm compactadas com o PostGIS. Ambas views utilizam informações embutidas em restrições das tabelas rasters. Como resultado as views catalogadas são sempre consistentes com os dados raster nas tabelas já que as restrições são impostas.

1. `raster_columns` esta view cataloga todas os tabelas de colunas rasters no seu banco de dados.
2. `raster_overviews` esta view cataloga todos as tabelas de colunas raster no seu banco de dados que servem como um panorama para uma tabela granulada melhor. Tabelas deste tipo são geradas quando você utiliza o interruptor `-l` durante o carregamento.

9.2.1 Catálogo de Colunas Raster

O `raster_columns` é um catálogo de todas as colunas de tabela raster no seu banco de dados que são do tipo raster. É uma view utilizando as restrições nas tabelas para que a informação seja sempre consistente, mesmo se você tiver restaurado uma tabela raster de um backup de outro banco de dados. As seguintes colunas existem no catálogo `raster_columns`.

Se você criou suas tabelas sem o carregador ou esqueceu de especificar a bandeira `-C` durante o carregamento, você pode forçar as restrições depois de usá-las de fato AddRasterConstraints para que o catálogo `raster_columns` registre as informações comuns sobre as tiles raster.

- `r_table_catalog` O banco de dados que a tabela está. Isto irá sempre ler o banco de dados atual.
- `r_table_schema` O esquema do banco de dados que o raster pertence.
- `r_table_name` raster table
- `r_raster_column` a coluna é a `r_table_name` tabela que é do tipo raster. Não há nada no PostGIS que previna múltiplas colunas raster por tabela, assim, é possível haver uma tabela raster listada várias vezes com uma coluna raster diferente pra cada uma.
- `srid` O identificador de referência espacial do raster. Deve ser uma entrada no Section 4.5.
- `scale_x` A escala entre coordenadas geométricas espaciais e pixel. Isto só está disponível se todas as tiles na coluna raster tiverem a mesma `scale_x` e esta restrição for aplicada. Recorra a ST_ScaleX para mais detalhes.
- `scale_y` A escala entre coordenadas geométricas espaciais e pixel. Isto só está disponível se todas as tiles na coluna raster tiverem a mesma `scale_y` e a restrição `scale_y` for aplicada. Recorra a ST_ScaleY para mais detalhes.
- `blocksize_x` A largura (número de pixels obliquamente) de cada raster tile. Recorrer aST_Width para mais detalhes.
- `blocksize_y` A largura (número de pixels para baixo) de cada raster tile. Recorrer a ST_Height para mais detalhes.

- `same_alignment` Uma booleana que é verdade se todos os rasters tiles têm o mesmo alinhamento. Recorrer a ST_SameAlignment para mais detalhes.
- `regular_blocking` Se a coluna raster possui a espacialidade única e cobre restrições tiles, o valor com ela se torna VERDADE. Senão, será FALSO.
- `num_bands` O número de bandas em cada tile do seu conjunto de raster. É a mesma informação da que é fornecida por ST_NumBands
- `pixel_types` Um arranjo definindo o tipo de pixel para cada banda. Você terá o mesmo número de elementos e bandas nesse arranjo. Os pixel_types são uns dos definidos em ST_BandPixelType.
- `nodata_values` Um arranjo de números preciso dobrados indicando o `nodata_value` para cada banda. Você terá o mesmo número de elementos e de bandas neste arranjo. Esses números definem o valor do pixel para cada banda que deveria ser ignorada para a maioria das operações. Uma informação parecida é fornecida por: ST_BandNoDataValue.
- `out_db` Um arranjo de bandeiras booleanas indicando se os dados das bandas rasters são mantidos de fora do banco de dados. Você terá o mesmo número de elementos e bandas neste arranjo.
- `extent` Isto é uma extensão de todas as filas raster no sua configuração raster. Se você planeja carregar mais dados que irão modificar a extensão de configuração, precisará executar a função DropRasterConstraints antes de carregar e então reaplicar as restrições com AddRasterConstraints depois carregar.
- `spatial_index` Uma booleana que é verdade se uma coluna raster possui um índice espacial.

9.2.2 Panoramas Raster

`raster_overviews` cataloga informação sobre as colunas de tabelas raster usadas para panoramas e informações adicionais sobre elas, que são úteis para saber quando usar os panoramas. As tabelas de panoramas são catalogadas em `raster_columns` e `raster_overviews`, porque elas são raster, mas também têm um propósito especial de serem uma caricatura de baixa resolução de uma tabela de alta resolução. Elas são geradas ao longo do lado da tabela raster principal quando você usa a troca `-l` no carregamento raster ou podem ser geradas manualmente, utilizando: AddOverviewConstraints.

Tabelas resumidas contêm as mesmas restrições que as outras tabelas raster bem como informações adicionais somente restrições específicas para panoramas.

Note
A informação em `raster_overviews` não duplica a informação em `raster_columns`. Se você precisa da informação sobre uma tabela panorama presente em `raster_columns`, você pode unir as `raster_overviews` e `raster_columns` para obter o conjunto completo de informações que precisa.

Duas razões principais para panoramas são:

1. Baixa resolução das tabelas de núcleo comumente usadas para um mapeamento de aproximação mais rápido.
2. Os cálculos são, geralmente, mais rápidos de serem feitos em si mesmos que a resolução mais alta de seus parentes, porque são relatos menores e cada pixel cobre mais território. Embora os cálculos não são tão atuais quanto as tabelas high-res que eles suportam, eles pode ser suficientes em vários cálculos da regra do polegar.

O catálogo `raster_overviews` contém as seguintes colunas de informação.

- `o_table_catalog` O banco de dados que o panorama está localizado. Isto sempre irá ler o banco de dados atual.
- `o_table_schema` O esquema do banco de dados que a tabela do panorama raster pertence.
- `o_table_name` nome da tabela de panorama raster
- `o_raster_column` a coluna raster na tabela panorama

- `r_table_catalog` O banco de dados que a tabela raster que este panorama está. Isto sempre lerá o banco de dados atual.
- `r_table_schema` O esquema do banco de dados da tabela raster que os serviços do panorama pertence.
- `r_table_name` tabela raster que este panorama fornece.
- `r_raster_column` a coluna raster que esta coluna panorama fornece.
- `overview_factor` - este é o nível da pirâmide da tabela panorama. Quanto maior o número menor a resolução da tabela. raster2pgsql se dada uma pasta de imagens, irá calcular panorama de cada arquivo de imagem e carregar separadamente. O nível 1 é assumido e sempre o arquivo original. Nível 2 terá que cada tile representa 4 do original. Então, por exemplo, se você tem uma pasta com arquivos de imagens de 5000x5000 pixel e você escolhe ordenar 125x125, para cada arquivo de imagem sua tabela base terá (5000*5000)/(125*125) records = 1600, your (l=2) `o_2` will have ceiling(1600/Power(2,2)) = 400 rows, your (l=3) `o_3` will have ceiling(1600/Power(2,3)) = 200 rows. Se seus pixels não são visíveis pelo tamanho das suas tiles, você pegará algumas tiles sobras (que não estão completamente cheias). Note que cada tile panorama gerada pelo raster2pgsql tem o meso número de pixels de seus pais, mas é de uma resolução menor onde cada pixel dele representa (Power(2,overview_factor) pixels do original).

9.3 Construindo Aplicações Personalizadas com o PostGIS Raster

The fact that PostGIS raster provides you with SQL functions to render rasters in known image formats gives you a lot of options for rendering them. For example you can use OpenOffice / LibreOffice for rendering as demonstrated in Rendering PostGIS Raster graphics with LibreOffice Base Reports. In addition you can use a wide variety of languages as demonstrated in this section.

9.3.1 PHP Exemplo Outputting usando ST_AsPNG em consenso co outras funções raster

Nesta seção, demonstraremos como usar o driver PHP PostgreSQL e a família ST_AsGDALRaster de funções para gerar banda 1,2,3 de um raster para um fluxo de solicitação PHP que pode ser inserido em uma img src html tag.

A consulta exemplo demonstra como combinar um conjunto de funções raster para apanhar todas as tiles que intersectam uma caixa delimitadora wgs 84 e então une com ST_Union as tiles intersectando retornando todas as bandas, transforma para projeção de usuário específico usando ST_Transform, e gera os resultados como um png usando ST_AsPNG.

Você poderia chamar o abaixo usando

```
http://mywebserver/test_raster.php?srid=2249
```

para obter a imagem raster no Massachusetts state plane feet.

```
<?php
/** contents of test_raster.php **/
$conn_str ='dbname=mydb host=localhost port=5432 user=myuser password=mypwd';
$dbconn = pg_connect($conn_str);
header('Content-Type: image/png');
/**If a particular projection was requested use it otherwise use mass state plane meters  ↩
    **/
if (!empty( $_REQUEST['srid'] ) && is_numeric( $_REQUEST['srid']) ){
                $input_srid = intval($_REQUEST['srid']);
}
else { $input_srid = 26986; }
/** The set bytea_output may be needed for PostgreSQL 9.0+, but not for 8.4 **/
$sql = "set bytea_output='escape';
SELECT ST_AsPNG(ST_Transform(
                        ST_AddBand(ST_Union(rast,1), ARRAY[ST_Union(rast,2),ST_Union(rast ↩
                            ,3)])
                                ,$input_srid) ) As new_rast
 FROM aerials.boston
        WHERE
```

```
         ST_Intersects(rast, ST_Transform(ST_MakeEnvelope(-71.1217, 42.227, -71.1210,  ↩
            42.218,4326),26986) )";
$result = pg_query($sql);
$row = pg_fetch_row($result);
pg_free_result($result);
if ($row === false) return;
echo pg_unescape_bytea($row[0]);
?>
```

9.3.2 ASP.NET C# Exemplo gerado usando ST_AsPNG em consenso com outras funções raster

Nesta seção, demonstraremos como usar o driver PHP PostgreSQL .NET driver e a família ST_AsGDALRaster de funções para gerar banda 1,2,3 de um raster para um fluxo de solicitação PHP que pode ser inserido em uma img src html tag.

Você precisará do driver npgsql .NET PostgreSQL para este exercício que pode ser obtido em: http://npgsql.projects.postgresql.org/- . Apenas faça o download e coloque na sua pasta ASP.NET bin e você estará pronto.

A consulta exemplo demonstra como combinar um conjunto de funções raster para apanhar todas as tiles que intersectam uma caixa delimitadora wgs 84 e então une com ST_Union as tiles intersectando retornando todas as bandas, transforma para projeção de usuário específico usando ST_Transform, e gera os resultados como um png usando ST_AsPNG.

Este é o mesmo exemplo de Section 9.3.1 exceto implementado em C#.

Você poderia chamar o abaixo usando

```
http://mywebserver/TestRaster.ashx?srid=2249
```

para obter a imagem raster no Massachusetts state plane feet.

```
-- web.config connection string section --
<connectionStrings>
    <add name="DSN"
        connectionString="server=localhost;database=mydb;Port=5432;User Id=myuser;password= ↩
            mypwd"/>
</connectionStrings
>
```

```
// Code for TestRaster.ashx
<%@ WebHandler Language="C#" Class="TestRaster" %>
using System;
using System.Data;
using System.Web;
using Npgsql;

public class TestRaster : IHttpHandler
{
        public void ProcessRequest(HttpContext context)
        {

                context.Response.ContentType = "image/png";
                context.Response.BinaryWrite(GetResults(context));

        }

        public bool IsReusable {
                get { return false; }
        }

        public byte[] GetResults(HttpContext context)
        {
                byte[] result = null;
```

```
                NpgsqlCommand command;
                string sql = null;
                int input_srid = 26986;
        try {
                    using (NpgsqlConnection conn = new NpgsqlConnection(System. ↩
                        Configuration.ConfigurationManager.ConnectionStrings["DSN"]. ↩
                        ConnectionString)) {
                            conn.Open();

                if (context.Request["srid"] != null)
                {
                    input_srid = Convert.ToInt32(context.Request["srid"]);
                }
                sql = @"SELECT ST_AsPNG(
                            ST_Transform(
                                        ST_AddBand(
                                    ST_Union(rast,1), ARRAY[ST_Union(rast,2),ST_Union(rast,3)])
                                                ,:input_srid) ) As new_rast
                            FROM aerials.boston
                                    WHERE
                                        ST_Intersects(rast,
                                        ST_Transform(ST_MakeEnvelope(-71.1217, 42.227,  ↩
                                            -71.1210, 42.218,4326),26986) )";
                            command = new NpgsqlCommand(sql, conn);
                command.Parameters.Add(new NpgsqlParameter("input_srid", input_srid));


                            result = (byte[]) command.ExecuteScalar();
                conn.Close();
                        }

                }
        catch (Exception ex)
        {
            result = null;
            context.Response.Write(ex.Message.Trim());
        }
                return result;
        }
}
```

9.3.3 O app console Java que gera a consulta raster como arquivo de imagem

Este é um exemplo de aplicativo console java que utiliza uma consulta que retorna uma imagem e gera um arquivo específico.

Você pode baixar os últimos drivers PostgreSQL JDBC de http://jdbc.postgresql.org/download.html

Você pode compilar o código seguinte usando um comando como:

```
set env CLASSPATH .:..\postgresql-9.0-801.jdbc4.jar
javac SaveQueryImage.java
jar cfm SaveQueryImage.jar Manifest.txt *.class
```

E chama da linha de comando com algo tipo

```
java -jar SaveQueryImage.jar "SELECT ST_AsPNG(ST_AsRaster(ST_Buffer(ST_Point(1,5),10, ' ↩
    quad_segs=2'),150, 150, '8BUI',100));" "test.png"
```

```
-- Manifest.txt --
Class-Path: postgresql-9.0-801.jdbc4.jar
Main-Class: SaveQueryImage
```

```
// Code for SaveQueryImage.java
import java.sql.Connection;
import java.sql.SQLException;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.io.*;

public class SaveQueryImage {
  public static void main(String[] argv) {
      System.out.println("Checking if Driver is registered with DriverManager.");

      try {
        //java.sql.DriverManager.registerDriver (new org.postgresql.Driver());
        Class.forName("org.postgresql.Driver");
      }
      catch (ClassNotFoundException cnfe) {
        System.out.println("Couldn't find the driver!");
        cnfe.printStackTrace();
        System.exit(1);
      }

      Connection conn = null;

      try {
        conn = DriverManager.getConnection("jdbc:postgresql://localhost:5432/mydb","myuser ↩
            ", "mypwd");
        conn.setAutoCommit(false);

        PreparedStatement sGetImg = conn.prepareStatement(argv[0]);

        ResultSet rs = sGetImg.executeQuery();

                FileOutputStream fout;
                try
                {
                        rs.next();
                        /** Output to file name requested by user **/
                        fout = new FileOutputStream(new File(argv[1]) );
                        fout.write(rs.getBytes(1));
                        fout.close();
                }
                catch(Exception e)
                {
                        System.out.println("Can't create file");
                        e.printStackTrace();
                }

        rs.close();
                sGetImg.close();
        conn.close();
      }
      catch (SQLException se) {
        System.out.println("Couldn't connect: print out a stack trace and exit.");
        se.printStackTrace();
        System.exit(1);
      }
  }
}
```

9.3.4 Use PLPython para excluir imagens via SQL

Esta é uma função plpython armazenada que cria um arquivo no diretório do servidor para cada relato. Requer que tenha instalado plpython. Deve funcionar bem com plpythonu e plpython3u.

```
CREATE OR REPLACE FUNCTION write_file (param_bytes bytea, param_filepath text)
RETURNS text
AS $$
f = open(param_filepath, 'wb+')
f.write(param_bytes)
return param_filepath
$$ LANGUAGE plpythonu;
```

```
--write out 5 images to the PostgreSQL server in varying sizes
-- note the postgresql daemon account needs to have write access to folder
-- this echos back the file names created;
 SELECT write_file(ST_AsPNG(
        ST_AsRaster(ST_Buffer(ST_Point(1,5),j*5, 'quad_segs=2'),150*j, 150*j, '8BUI',100)),
         'C:/temp/slices'|| j || '.png')
         FROM generate_series(1,5) As j;

     write_file
---------------------
 C:/temp/slices1.png
 C:/temp/slices2.png
 C:/temp/slices3.png
 C:/temp/slices4.png
 C:/temp/slices5.png
```

9.3.5 Rasters de saída com PSQL

Infelizmente o PSQL não tem facilidade em usar funcionalidade embutida para binários gerados. Isto é um pequeno hack no legado PostgreSQL de suporte de objetos grandes. Para usar, primeiro lance sua linha de comando psql conectada no seu banco de dados.

Diferente da aproximação python, esta cria o arquivo no seu computador local.

```
SELECT oid, lowrite(lo_open(oid, 131072), png) As num_bytes
 FROM
 ( VALUES (lo_create(0),
   ST_AsPNG( (SELECT rast FROM aerials.boston WHERE rid=1) )
  ) ) As v(oid,png);
-- you'll get an output something like --
   oid   | num_bytes
---------+-----------
 2630819 |     74860

-- next note the oid and do this replacing the c:/test.png to file path location
-- on your local computer
 \lo_export 2630819 'C:/temp/aerial_samp.png'

-- this deletes the file from large object storage on db
SELECT lo_unlink(2630819);
```

Chapter 10

Referência Raster

As funções abaixo são as que um usuário do PostGIS Raster vai precisar e que estão disponíveis no momento no PostGIS Raster. Existem outras funções que precisam das funções suporte para os objetos raster que não são de uso geral.

`raster` é um novo tipo PostGIS para armazenar e analizar dados raster.

Para carregar rasters de arquivos raster, por favor, vá para Section 9.1

Para os exemplos nessa referência, nós usaremos uma table raster de rasters simulados - Formados com o seguinte código

```
CREATE TABLE dummy_rast(rid integer, rast raster);
INSERT INTO dummy_rast(rid, rast)
VALUES (1,
('01' -- little endian (uint8 ndr)
||
'0000' -- version (uint16 0)
||
'0000' -- nBands (uint16 0)
||
'0000000000000040' -- scaleX (float64 2)
||
'0000000000000840' -- scaleY (float64 3)
||
'000000000000E03F' -- ipX (float64 0.5)
||
'000000000000E03F' -- ipY (float64 0.5)
||
'0000000000000000' -- skewX (float64 0)
||
'0000000000000000' -- skewY (float64 0)
||
'00000000' -- SRID (int32 0)
||
'0A00' -- width (uint16 10)
||
'1400' -- height (uint16 20)
)::raster
),
-- Raster: 5 x 5 pixels, 3 bands, PT_8BUI pixel type, NODATA = 0
(2,  ('01000003009A9999999999A93F9A9999999999A9BF000000E02B274A' ||
'410000000077195641000000000000000000000000000000000000 ↩
    FFFFFFFF050005000400FDFEFDFEFEFDFEFEFDF9FAFEF' ||
' ↩
    EFCF9FBFDFEFEFDFCFAFEFEFE04004E627AADD16076B4F9FE6370A9F5FE59637AB0E54F5861708704004656648 7A15060
    ')::raster);
```

10.1 Tipos de suporte de dados raster

10.1.1 geomval

geomval — Um tipo de dado espacial com dois campos - geom (possuindo objeto geométrico) e val (possuindo um valor de pixel de precisão dupla de uma banda raster).

Descrição

geomval é uma mistura de tipo de dados que consiste em um objeto de geometria referenciado pelo campo .geom e val, um valor de precisão dupla que representa o valor do pixel em uma localização específica de geometria em uma banda raster. É usado por ST_DumpAsPolygon e a família de interseção raster de funções como um tipo de saída para explodir uma banda raster em polígonos.

Veja também

Section 12.6

10.1.2 addbandarg

addbandarg — Um tipo composto usado como entrada na função ST_AddBand definindo os atributos e valor inicial da nova banda.

Descrição

Um tipo composto usado como entrada na função ST_AddBand definindo os atributos e valor inicial da nova banda.

index **integer** Valor de 1-base indicando a posição onde a nova banda será adicionada no meio das bandas do raster. Se NULO, a nova banda será adicionada no fim das bandas do raster.

pixeltype **text** tipo do pixel da nova banda. Um dos tipos de pixel definidos como descrito em: ST_BandPixelType.

initialvalue **double precision** Valor inicial que todos os pixels da nova banda serão definidos.

nodataval **double precision** Valor NODATA da nova banda. Se NULA, a nova banda terá uma valor NODATA assinado.

Veja também

ST_AddBand

10.1.3 rastbandarg

rastbandarg — Um tipo composto para usar quando for preciso expressar um raster e um índice de banda desse raster.

Descrição

Um tipo composto para usar quando for preciso expressar um raster e um índice de banda desse raster.

rast **raster** O raster em questão/

nband **integer** Valor 1-base indicando a banda do raster

Veja também

Funções retorno de mapa algébrico embutido

10.1.4 raster

raster — raster spatial data type.

Descrição

raster is a spatial data type used to represent raster data such as those imported from JPEGs, TIFFs, PNGs, digital elevation models. Each raster has 1 or more bands each having a set of pixel values. Rasters can be georeferenced.

Note
Requer que o PostGIS esteja compilado com o suporte GDAL. Os rasters, atualmente, podem ser convertidos implicitamente para geometria, mas a conversão retorna a ST_ConvexHull do raster. Este auto casting pode ser removido em futuro próximo, então não confie muito nisto.

Comportamento Casting

Essa seção lista os casts automáticos bem como os explícitos permitidos para esse tipo de dados

Cast To	Comportamento
geometria	automático

Veja também

Chapter 10

10.1.5 reclassarg

reclassarg — Um tipo composto usado como entrada dentro da função ST_Reclass definindo o comportamento da reclassificação.

Descrição

Um tipo composto usado como entrada dentro da função ST_Reclass definindo o comportamento da reclassificação.

nband **integer** O número banda para banda para reclassificar.

reclassexpr **text** expressão de variação consistindo em mapeamentos range:map_range delimitados por vírgulas. : para definir mapeamento que esclarece como mapear valores antigos de banda para novos. (means >,) significa menor que,] < ou igual, [significa > ou igual

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

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

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

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

(notação é opcional então a-b significa o mesmo que (a-b)

pixeltype **text** Um dos tipos de pixel definidos como descrito em: ST_BandPixelType

nodataval **double precision** Valor para tratar como sem dados. Para saídas de imagens que suportam transparência, essas serão em branco.

Exemplo: Reclassificar banda 2 como um 8BUI onde 255 é o valor sem dados

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

Exemplo: Reclassificar banda 1 como um 1BB e nenhum valor sem dados definido

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

Veja também

ST_Reclass

10.1.6 summarystats

summarystats — Um tipo composto retornado pelas funções ST_SummaryStats e ST_SummaryStatsAgg.

Descrição

Um tipo composto retornado pelas funções ST_SummaryStats e ST_SummaryStatsAgg.

count **integer** Número de pixels contados para as estatísticas resumo.

sum **double precision** Resumo de todos os valores contados de pixels.

mean **double precision** Significado aritmético de todos os valores de pixels.

stddev **double precision** Divergência parão de todos os valores contados de pixels.

min **double precision** Valor mínimo dos valores dos pixeis contados.

max **double precision** Valor máximo dos valores dos pixeis contados.

Veja também

ST_SummaryStats, ST_SummaryStatsAgg

10.1.7 unionarg

unionarg — Um tipo composto usado como entrada dentro da função ST_Union definindo as bandas a serem processadas e o comportamento da operação UNIÃO.

Descrição

Um tipo composto usado como entrada dentro da função ST_Union definindo as bandas a serem processadas e o comportamento da operação UNIÃO.

nband **integer** Valor 1-baseado indicando a banda de cada raster de entrada a ser processado.

uniontype **text** Tipo de operação de UNIÃO. Um dos tipos definidos como descritos em ST_Union.

Veja também

ST_Union

10.2 Gerenciamento Raster

10.2.1 AddRasterConstraints

AddRasterConstraints — Adds raster constraints to a loaded raster table for a specific column that constrains spatial ref, scaling, blocksize, alignment, bands, band type and a flag to denote if raster column is regularly blocked. The table must be loaded with data for the constraints to be inferred. Returns true if the constraint setting was accomplished and issues a notice otherwise.

Synopsis

boolean **AddRasterConstraints**(name rasttable, name rastcolumn, boolean srid=true, boolean scale_x=true, boolean scale_y=true, boolean blocksize_x=true, boolean blocksize_y=true, boolean same_alignment=true, boolean regular_blocking=false, boolean num_bands=true , boolean pixel_types=true , boolean nodata_values=true , boolean out_db=true , boolean extent=true);
boolean **AddRasterConstraints**(name rasttable, name rastcolumn, text[] VARIADIC constraints);
boolean **AddRasterConstraints**(name rastschema, name rasttable, name rastcolumn, text[] VARIADIC constraints);
boolean **AddRasterConstraints**(name rastschema, name rasttable, name rastcolumn, boolean srid=true, boolean scale_x=true, boolean scale_y=true, boolean blocksize_x=true, boolean blocksize_y=true, boolean same_alignment=true, boolean regular_blocking=fa
boolean num_bands=true, boolean pixel_types=true, boolean nodata_values=true , boolean out_db=true , boolean extent=true);

Descrição

Gera restrições em uma coluna raster que são usadas para expor informação no catálogo raster `raster_columns`. O `rastschema` é o nome da tabela esquema que a tabela está. O `srid` deve ser um valor inteiro referência a uma entrada na tabela SPATIAL_REF_SYS.

`raster2pgsql` o carregador usa esta função para registrar tabelas raster

Valida nomes restritos para passar: recorra a Section 9.2.1 para mais detalhes.

- `blocksize` coloca X e Y blocksize
- `blocksize_x` coloca tile X (largura em pixeis de cada tile)
- `blocksize_y` coloca tile Y (altura em pixeis de cada tile)
- `extent` calcula a extensão da tabela toda e aplica restrições, todos os rasters devem estar dentro da extensão
- `num_bands` número de bandas
- `pixel_types` lê arranjo de tipos de pixeis para cada banda garantir que todas as bandas n tenham o mesmo tipo de pixel
- `regular_blocking` espacialmente único (dois rasters não podem ser espacialmente iguais) e restrições de tile de cobertura (raster é alinhado a uma cobertura)
- `same_alignment` ensures they all have same alignment meaning any two tiles you compare will return true for. Refer to ST_SameAlignment.
- `srid` assegura que todos tenham o mesmo srid
- Mais -- qualquer um listado como entrada dentro das funções acima

Note
Esta função infere as restrições dos dados já presentes na tabela. Assim como para que ela funcione, você deve criar a coluna raster primeiro e então carregá-la com dados.

Note
Se você precisar carregar mais dados nas suas tabelas depois de ter aplicado suas restrições, talvez queira executar as DropRasterConstraints se a extensão dos seus dados mudou.

Disponibilidade: 2.0.0

Exemplos: Aplica todas as restrições possíveis em uma coluna baseada em dados

```
CREATE TABLE myrasters(rid SERIAL primary key, rast raster);
INSERT INTO myrasters(rast)
SELECT ST_AddBand(ST_MakeEmptyRaster(1000, 1000, 0.3, -0.3, 2, 2, 0, 0,4326), 1, '8BSI':: ↩
    text, -129, NULL);

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


-- verify if registered correctly in the raster_columns view --
SELECT srid, scale_x, scale_y, blocksize_x, blocksize_y, num_bands, pixel_types,  ↩
    nodata_values
    FROM raster_columns
    WHERE r_table_name = 'myrasters';

 srid | scale_x | scale_y | blocksize_x | blocksize_y | num_bands | pixel_types|  ↩
     nodata_values
------+---------+---------+-------------+-------------+-----------+-------------+--------------- ↩

 4326 |       2 |       2 |        1000 |        1000 |         1 | {8BSI}      | {0}
```

Exemplos: Aplica uma única restrição

```
CREATE TABLE public.myrasters2(rid SERIAL primary key, rast raster);
INSERT INTO myrasters2(rast)
SELECT ST_AddBand(ST_MakeEmptyRaster(1000, 1000, 0.3, -0.3, 2, 2, 0, 0,4326), 1, '8BSI':: ↩
    text, -129, NULL);

SELECT AddRasterConstraints('public'::name, 'myrasters2'::name, 'rast'::name,' ↩
    regular_blocking', 'blocksize');
-- get notice--
NOTICE:  Adding regular blocking constraint
NOTICE:  Adding blocksize-X constraint
NOTICE:  Adding blocksize-Y constraint
```

Veja também

Section 9.2.1, ST_AddBand, ST_MakeEmptyRaster, DropRasterConstraints, ST_BandPixelType, ST_SRID

10.2.2 DropRasterConstraints

DropRasterConstraints — Derruba as restrições raster PostGIS que se referem a uma tabela de coluna raster. É útil se você precisar recarregar dados ou atualizar os dados da sua coluna raster.

Synopsis

boolean **DropRasterConstraints**(name rasttable, name rastcolumn, boolean srid, boolean scale_x, boolean scale_y, boolean blocksize_x, boolean blocksize_y, boolean same_alignment, boolean regular_blocking, boolean num_bands=true, boolean pixel_types=t
boolean nodata_values=true, boolean out_db=true , boolean extent=true);
boolean **DropRasterConstraints**(name rastschema, name rasttable, name rastcolumn, boolean srid=true, boolean scale_x=true, boolean scale_y=true, boolean blocksize_x=true, boolean blocksize_y=true, boolean same_alignment=true, boolean regular_blocking=fa
boolean num_bands=true, boolean pixel_types=true, boolean nodata_values=true, boolean out_db=true , boolean extent=true);
boolean **DropRasterConstraints**(name rastschema, name rasttable, name rastcolumn, text[] constraints);

Descrição

Derruba as restrições raster PostGIS que se referem a uma tabela de coluna raster que foram adicionadas pela AddRasterConstraints. É útil se você precisar recarregar dados ou atualizar os dados da sua coluna raster. Não é necessário fazer isso se quiser livrar-se de uma tabela ou coluna raster.

Para derrubar uma tabela raster use o padrão

```
DROP TABLE mytable
```

Para derrubar uma coluna raster e deixar o resto da tabela, use o SQL padrão

```
ALTER TABLE mytable DROP COLUMN rast
```

a tabela desparecerá do catálogo `raster_columns` se a coluna ou tabela for derrubada. Entretanto, se somente as restrições forem derrubadas, a coluna raster continuará sendo listada no catálogo `raster_columns`, mas não haverá nenhuma outra informação sobre isso à parte do nome da coluna e da tabela.

Disponibilidade: 2.0.0

Exemplos

```
SELECT DropRasterConstraints ('myrasters','rast');
----RESULT output ---
t

-- verify change in raster_columns --
SELECT srid, scale_x, scale_y, blocksize_x, blocksize_y, num_bands, pixel_types,  ↩
    nodata_values
    FROM raster_columns
    WHERE r_table_name = 'myrasters';

 srid | scale_x | scale_y | blocksize_x | blocksize_y | num_bands | pixel_types|  ↩
     nodata_values
------+---------+---------+-------------+-------------+-----------+-------------+--------------- ↩

    0 |         |         |             |             |           |             |
```

Veja também

AddRasterConstraints

10.2.3 AddOverviewConstraints

AddOverviewConstraints — Marca uma coluna raster como sendo um resumo de outra.

Synopsis

boolean **AddOverviewConstraints**(name ovschema, name ovtable, name ovcolumn, name refschema, name reftable, name refcolumn, int ovfactor);
boolean **AddOverviewConstraints**(name ovtable, name ovcolumn, name reftable, name refcolumn, int ovfactor);

Descrição

Adiciona restrições em uma coluna raster que são usadas para expor informações no catálogo `raster_overviews` raster.

O parâmetro `ovfactor` representa a escala multiplicadora na coluna resumo: fatores resumo mais altos possuem uma resolução menor.

Quando os parâmetros `ovschema` e `refschema` são omitidos, a primeira tabela encontrada escaneando o `search_path` será utilizada.

Disponibilidade: 2.0.0

Exemplos

```
CREATE TABLE res1 AS SELECT
ST_AddBand(
  ST_MakeEmptyRaster(1000, 1000, 0, 0, 2),
  1, '8BSI'::text, -129, NULL
) r1;

CREATE TABLE res2 AS SELECT
ST_AddBand(
  ST_MakeEmptyRaster(500, 500, 0, 0, 4),
  1, '8BSI'::text, -129, NULL
) r2;

SELECT AddOverviewConstraints('res2', 'r2', 'res1', 'r1', 2);

-- verify if registered correctly in the raster_overviews view --
SELECT o_table_name ot, o_raster_column oc,
       r_table_name rt, r_raster_column rc,
       overview_factor f
FROM raster_overviews WHERE o_table_name = 'res2';
  ot  | oc |  rt  | rc | f
------+----+------+----+---
 res2 | r2 | res1 | r1 | 2
(1 row)
```

Veja também

Section 9.2.2, DropOverviewConstraints, ST_CreateOverview, AddRasterConstraints

10.2.4 DropOverviewConstraints

DropOverviewConstraints — Desmarca uma coluna raster de ser um resumo de outra.

Synopsis

boolean **DropOverviewConstraints**(name ovschema, name ovtable, name ovcolumn);
boolean **DropOverviewConstraints**(name ovtable, name ovcolumn);

Descrição

Remove as restrições de uma coluna raster usadas para apresentá-la como um resumo de outra no catálogo `raster_overviews` raster.

Quando o parâmetro `ovschema` é omitido, a primeira tabela encontradas escaneando o `search_path` será utilizada.

Disponibilidade: 2.0.0

Veja também

Section 9.2.2, AddOverviewConstraints, DropRasterConstraints

10.2.5 PostGIS_GDAL_Version

PostGIS_GDAL_Version — Relata a versão da biblioteca GDAL em uso pelo PostGIS

Synopsis

text **PostGIS_GDAL_Version**();

Descrição

Relata a versão da biblioteca em uso pelo PostGIS. Também irá verificar e reportar se GDAL pode encontrar os dados de seus arquivos.

Exemplos

```
SELECT PostGIS_GDAL_Version();
       postgis_gdal_version
-----------------------------------
 GDAL 1.11dev, released 2013/04/13
```

Veja também

postgis.gdal_datapath

10.2.6 PostGIS_Raster_Lib_Build_Date

PostGIS_Raster_Lib_Build_Date — Relata a data da biblioteca raster construída completa.

Synopsis

text **PostGIS_Raster_Lib_Build_Date**();

Descrição

Relata a data de construção raster

Exemplos

```
SELECT PostGIS_Raster_Lib_Build_Date();
postgis_raster_lib_build_date
-----------------------------
2010-04-28 21:15:10
```

Veja também

PostGIS_Raster_Lib_Version

10.2.7 PostGIS_Raster_Lib_Version

PostGIS_Raster_Lib_Version — Relata a versão raster completa e constrói informações de configuração.

Synopsis

text **PostGIS_Raster_Lib_Version**();

Descrição

Relata a versão raster completa e constrói informações de configuração.

Exemplos

```
SELECT PostGIS_Raster_Lib_Version();
postgis_raster_lib_version
--------------------------
 2.0.0
```

Veja também

PostGIS_Lib_Version

10.2.8 ST_GDALDrivers

ST_GDALDrivers — Returns a list of raster formats supported by PostGIS through GDAL. Only those formats with can_write=True can be used by ST_AsGDALRaster

Synopsis

setof record **ST_GDALDrivers**(integer OUT idx, text OUT short_name, text OUT long_name, text OUT can_read, text OUT can_write, text OUT create_options);

Descrição

Returns a list of raster formats short_name,long_name and creator options of each format supported by GDAL. Use the short_name as input in the `format` parameter of ST_AsGDALRaster. Options vary depending on what drivers your libgdal was compiled with. `create_options` returns an xml formatted set of CreationOptionList/Option consisting of name and optional `type`, `description` and set of `VALUE` for each creator option for the specific driver.

Changed: 2.5.0 - add can_read and can_write columns.

Alterações: 2.0.6, 2.1.3 - por padrão nenhum driver é ativado, a menos que GUC ou a variável ambiental gdal_enabled_drivers estejam colocadas.

Disponibilidade: 2.0.0 - requer GDAL >= 1.6.0.

Exemplos: Lista de Dispositivos

```
SET postgis.gdal_enabled_drivers = 'ENABLE_ALL';
SELECT short_name, long_name, can_write
FROM st_gdaldrivers()
ORDER BY short_name;

   short_name    |                          long_name                          | can_write
-----------------+-----------------------------------------------------------+-----------
 AAIGrid         | Arc/Info ASCII Grid                                       | t
 ACE2            | ACE2                                                      | f
 ADRG            | ARC Digitized Raster Graphics                             | f
 AIG             | Arc/Info Binary Grid                                      | f
 AirSAR          | AirSAR Polarimetric Image                                 | f
 ARG             | Azavea Raster Grid format                                 | t
 BAG             | Bathymetry Attributed Grid                                | f
 BIGGIF          | Graphics Interchange Format (.gif)                        | f
 BLX             | Magellan topo (.blx)                                      | t
 BMP             | MS Windows Device Independent Bitmap                      | f
 BSB             | Maptech BSB Nautical Charts                               | f
 PAux            | PCI .aux Labelled                                         | f
 PCIDSK          | PCIDSK Database File                                      | f
 PCRaster        | PCRaster Raster File                                      | f
 PDF             | Geospatial PDF                                            | f
 PDS             | NASA Planetary Data System                                | f
 PDS4            | NASA Planetary Data System 4                              | t
 PLMOSAIC        | Planet Labs Mosaics API                                   | f
 PLSCENES        | Planet Labs Scenes API                                    | f
 PNG             | Portable Network Graphics                                 | t
 PNM             | Portable Pixmap Format (netpbm)                           | f
 PRF             | Racurs PHOTOMOD PRF                                       | f
 R               | R Object Data Store                                       | t
 Rasterlite      | Rasterlite                                                | t
 RDA             | DigitalGlobe Raster Data Access driver                    | f
 RIK             | Swedish Grid RIK (.rik)                                   | f
 RMF             | Raster Matrix Format                                      | f
 ROI_PAC         | ROI_PAC raster                                            | f
 RPFTOC          | Raster Product Format TOC format                          | f
 RRASTER         | R Raster                                                  | f
 RS2             | RadarSat 2 XML Product                                    | f
 RST             | Idrisi Raster A.1                                         | t
 SAFE            | Sentinel-1 SAR SAFE Product                               | f
 SAGA            | SAGA GIS Binary Grid (.sdat, .sg-grd-z)                   | t
 SAR_CEOS        | CEOS SAR Image                                            | f
 SDTS            | SDTS Raster                                               | f
 SENTINEL2       | Sentinel 2                                                | f
 SGI             | SGI Image File Format 1.0                                 | f
```

```
 SNODAS          | Snow Data Assimilation System                                | f
 SRP             | Standard Raster Product (ASRP/USRP)                          | f
 SRTMHGT         | SRTMHGT File Format                                          | t
 Terragen        | Terragen heightfield                                         | f
 TIL             | EarthWatch .TIL                                              | f
 TSX             | TerraSAR-X Product                                           | f
 USGSDEM         | USGS Optional ASCII DEM (and CDED)                           | t
 VICAR           | MIPL VICAR file                                              | f
 VRT             | Virtual Raster                                               | t
 WCS             | OGC Web Coverage Service                                     | f
 WMS             | OGC Web Map Service                                          | t
 WMTS            | OGC Web Map Tile Service                                     | t
 XPM             | X11 PixMap Format                                            | t
 XYZ             | ASCII Gridded XYZ                                            | t
 ZMap            | ZMap Plus Grid                                               | t
```

Exemplo: Lista de opções para cada dispositivo

```
-- Output the create options XML column of JPEG as a table  --
-- Note you can use these creator options in ST_AsGDALRaster options argument
SELECT (xpath('@name', g.opt))[1]::text As oname,
       (xpath('@type', g.opt))[1]::text As otype,
       (xpath('@description', g.opt))[1]::text As descrip
FROM (SELECT unnest(xpath('/CreationOptionList/Option', create_options::xml)) As opt
FROM  st_gdaldrivers()
WHERE short_name = 'JPEG') As g;

       oname        |  otype  |      descrip
--------------------+---------+-------------------------------------------------
 PROGRESSIVE        | boolean | whether to generate a progressive JPEG
 QUALITY            | int     | good=100, bad=0, default=75
 WORLDFILE          | boolean | whether to geneate a worldfile
 INTERNAL_MASK      | boolean | whether to generate a validity mask
 COMMENT            | string  | Comment
 SOURCE_ICC_PROFILE | string  | ICC profile encoded in Base64
 EXIF_THUMBNAIL     | boolean | whether to generate an EXIF thumbnail(overview).
                                By default its max dimension will be 128
 THUMBNAIL_WIDTH    | int     | Forced thumbnail width
 THUMBNAIL_HEIGHT   | int     | Forced thumbnail height
(9 rows)
```

```
-- raw xml output for creator options for GeoTiff --
SELECT create_options
FROM st_gdaldrivers()
WHERE short_name = 'GTiff';

<CreationOptionList>
    <Option name="COMPRESS" type="string-select">
        <Value
>NONE</Value>
        <Value
>LZW</Value>
        <Value
>PACKBITS</Value>
        <Value
>JPEG</Value>
        <Value
>CCITTRLE</Value>
        <Value
>CCITTFAX3</Value>
```

```
        <Value
>CCITTFAX4</Value>
        <Value
>DEFLATE</Value>
    </Option>
    <Option name="PREDICTOR" type="int" description="Predictor Type"/>
    <Option name="JPEG_QUALITY" type="int" description="JPEG quality 1-100" default="75"/>
    <Option name="ZLEVEL" type="int" description="DEFLATE compression level 1-9" default ↩
        ="6"/>
    <Option name="NBITS" type="int" description="BITS for sub-byte files (1-7), sub-uint16  ↩
        (9-15), sub-uint32 (17-31)"/>
    <Option name="INTERLEAVE" type="string-select" default="PIXEL">
        <Value
>BAND</Value>
        <Value
>PIXEL</Value>
    </Option>
    <Option name="TILED" type="boolean" description="Switch to tiled format"/>
    <Option name="TFW" type="boolean" description="Write out world file"/>
    <Option name="RPB" type="boolean" description="Write out .RPB (RPC) file"/>
    <Option name="BLOCKXSIZE" type="int" description="Tile Width"/>
    <Option name="BLOCKYSIZE" type="int" description="Tile/Strip Height"/>
    <Option name="PHOTOMETRIC" type="string-select">
        <Value
>MINISBLACK</Value>
        <Value
>MINISWHITE</Value>
        <Value
>PALETTE</Value>
        <Value
>RGB</Value>
        <Value
>CMYK</Value>
        <Value
>YCBCR</Value>
        <Value
>CIELAB</Value>
        <Value
>ICCLAB</Value>
        <Value
>ITULAB</Value>
    </Option>
    <Option name="SPARSE_OK" type="boolean" description="Can newly created files have  ↩
        missing blocks?" default="FALSE"/>
    <Option name="ALPHA" type="boolean" description="Mark first extrasample as being alpha ↩
        "/>
    <Option name="PROFILE" type="string-select" default="GDALGeoTIFF">
        <Value
>GDALGeoTIFF</Value>
        <Value
>GeoTIFF</Value>
        <Value
>BASELINE</Value>
    </Option>
    <Option name="PIXELTYPE" type="string-select">
        <Value
>DEFAULT</Value>
        <Value
>SIGNEDBYTE</Value>
    </Option>
    <Option name="BIGTIFF" type="string-select" description="Force creation of BigTIFF file ↩
        ">
```

```
        <Value
>YES</Value>
        <Value
>NO</Value>
        <Value
>IF_NEEDED</Value>
        <Value
>IF_SAFER</Value>
    </Option>
    <Option name="ENDIANNESS" type="string-select" default="NATIVE" description="Force  ↩
        endianness of created file. For DEBUG purpose mostly">
        <Value
>NATIVE</Value>
        <Value
>INVERTED</Value>
        <Value
>LITTLE</Value>
        <Value
>BIG</Value>
    </Option>
    <Option name="COPY_SRC_OVERVIEWS" type="boolean" default="NO" description="Force copy  ↩
        of overviews of source dataset (CreateCopy())"/>
</CreationOptionList
>

-- Output the create options XML column for GTiff as a table  --
SELECT (xpath('@name', g.opt))[1]::text As oname,
       (xpath('@type', g.opt))[1]::text As otype,
       (xpath('@description', g.opt))[1]::text As descrip,
       array_to_string(xpath('Value/text()', g.opt),', ') As vals
FROM (SELECT unnest(xpath('/CreationOptionList/Option', create_options::xml)) As opt
FROM  st_gdaldrivers()
WHERE short_name = 'GTiff') As g;

       oname        |     otype     |                                descrip  ↩
                                    |                                      vals
--------------------+---------------+-------------------------------------------------------------------------

 COMPRESS           | string-select |  ↩
                                                                          | NONE, LZW,  ↩
    PACKBITS, JPEG, CCITTRLE, CCITTFAX3, CCITTFAX4, DEFLATE
 PREDICTOR          | int           | Predictor Type  ↩
                                                        |
 JPEG_QUALITY       | int           | JPEG quality 1-100  ↩
                                                       |
 ZLEVEL             | int           | DEFLATE compression level 1-9  ↩
                                                  |
 NBITS              | int           | BITS for sub-byte files (1-7), sub-uint16 (9-15), sub ↩
    -uint32 (17-31) |
 INTERLEAVE         | string-select |  ↩
                                                                          | BAND, PIXEL
 TILED              | boolean       | Switch to tiled format  ↩
                                                     |
 TFW                | boolean       | Write out world file  ↩
                                                      |
 RPB                | boolean       | Write out .RPB (RPC) file  ↩
                                                    |
 BLOCKXSIZE         | int           | Tile Width  ↩
                                                          |
 BLOCKYSIZE         | int           | Tile/Strip Height  ↩
                                                        |
 PHOTOMETRIC        | string-select |  ↩
```

```
                                                                    | MINISBLACK,  ↩
    MINISWHITE, PALETTE, RGB, CMYK, YCBCR, CIELAB, ICCLAB, ITULAB
 SPARSE_OK          | boolean       | Can newly created files have missing blocks?  ↩
                              |
 ALPHA              | boolean       | Mark first extrasample as being alpha  ↩
                             |
 PROFILE            | string-select |  ↩
                                                                    | GDALGeoTIFF,  ↩
    GeoTIFF, BASELINE
 PIXELTYPE          | string-select |  ↩
                                                                    | DEFAULT,  ↩
    SIGNEDBYTE
 BIGTIFF            | string-select | Force creation of BigTIFF file  ↩
                                          | YES, NO, IF_NEEDED, IF_SAFER
 ENDIANNESS         | string-select | Force endianness of created file. For DEBUG purpose  ↩
    mostly            | NATIVE, INVERTED, LITTLE, BIG
 COPY_SRC_OVERVIEWS | boolean       | Force copy of overviews of source dataset (CreateCopy ↩
    ())             |
(19 rows)
```

Veja também

ST_AsGDALRaster, ST_SRID, postgis.gdal_enabled_drivers

10.2.9 ST_Count

ST_Count — Generates a set of vector contours from the provided raster band, using the GDAL contouring algorithm.

Synopsis

raster **ST_TPI**(raster rast, integer nband, raster customextent, text pixeltype="32BF" , boolean interpolate_nodata=FALSE);

Descrição

Generates a set of vector contours from the provided raster band, using the GDAL contouring algorithm.

When the `fixed_levels` parameter is a non-empty array, the `level_interval` and `level_base` parameters are ignored.

Return values are a set of records with the following attributes:

geomval The geometry of the contour line.

id A unique identifier given to the contour line by GDAL.

ST_Value The raster value the line represents. For an elevation DEM input, this would be the elevation of the output contour.

Disponibilidade: 2.2.0

Exemplo

```
WITH c AS (
SELECT (ST_Contour(rast, 1, fixed_levels =
> ARRAY[100.0, 200.0, 300.0])).*
FROM dem_grid WHERE rid = 1
)
SELECT st_astext(geom), id, value
FROM c;
```

Veja também

ST_MakeEmptyRaster

10.2.10 ST_MakeEmptyRaster

ST_MakeEmptyRaster — Interpolates a gridded surface based on an input set of 3-d points, using the X- and Y-values to position the points on the grid and the Z-value of the points as the surface elevation.

Synopsis

bytea **ST_AsGDALRaster**(raster rast, text format, text[] options=NULL, integer srid=sameassource);

Descrição

Interpolates a gridded surface based on an input set of 3-d points, using the X- and Y-values to position the points on the grid and the Z-value of the points as the surface elevation. There are five interpolation algorithms available: inverse distance, inverse distance nearest-neighbor, moving average, nearest neighbor, and linear interpolation. See the gdal_grid documentation for more details on the algorithms and their parameters. For more information on how interpolations are calculated, see the GDAL grid tutorial.

Input parameters are:

input_points The points to drive the interpolation. Any geometry with Z-values is acceptable, all points in the input will be used.

algorithm_options A string defining the algorithm and algorithm options, in the format used by gdal_grid. For example, for an inverse-distance interpolation with a smoothing of 2, you would use "invdist:smoothing=2.0"

template A raster template to drive the geometry of the output raster. The width, height, pixel size, spatial extent and pixel type will be read from this template.

template_band_num By default the first band in the template raster is used to drive the output raster, but that can be adjusted with this parameter.

Disponibilidade: 2.2.0

Exemplo

```
SELECT ST_InterpolateRaster(
    'MULTIPOINT(10.5 9.5 1000, 11.5 8.5 1000, 10.5 8.5 500, 11.5 9.5 500)'::geometry,
    'invdist:smoothing:2.0',
    ST_AddBand(ST_MakeEmptyRaster(200, 400, 10, 10, 0.01, -0.005, 0, 0), '16BSI')
)
```

Veja também

ST_Count

10.2.11 UpdateRasterSRID

UpdateRasterSRID — Altera o SRID de todos os rasters na coluna e tabela do usuário especificado.

Synopsis

raster **UpdateRasterSRID**(name schema_name, name table_name, name column_name, integer new_srid);
raster **UpdateRasterSRID**(name table_name, name column_name, integer new_srid);

Descrição

Altera o SRID de todos os rasters na coluna e tabela do usuário especificado. A função irá derrubar todas as restrições de colunas apropriadas (extensão, alinhamento e SRID) antes de modificar o SRID dos rasters específicos da coluna.

Note
Os dados (banda valores pixel) dos rasters não são mexidos por esta função. Somente os metadados do restar são alterados.

Disponibilidade: 2.1.0

Veja também

UpdateGeometrySRID

10.2.12 ST_CreateOverview

ST_CreateOverview — Cria uma resolução de versão reduzida de uma dada cobertura raster.

Synopsis

regclass **ST_CreateOverview**(regclass tab, name col, int factor, text algo='NearestNeighbor');

Descrição

Cria uma tabela panorama com resampled tiles da tabela fonte. AS tiles de saída terão o mesmo tamanho das de entrada e cobrirão a mesma extensão espacial com uma resolução mais baixa (tamanho do pixel será 1/`factor` do original em ambas direções).

A tabela panorama se tronará disponível no catálogo `raster_overviews` e terá restrições raster executadas.

As opções de algorítimo são: 'NearestNeighbor', 'Bilinear', 'Cubic', 'CubicSpline', e 'Lanczos'. Recorra a: GDAL Warp resampling methods para mais detalhes.

Disponibilidade: 2.2.0

Exemplo

Output to generally better quality but slower to product format

```
SELECT ST_CreateOverview('mydata.mytable'::regclass, 'rast', 2, 'Lanczos');
```

Output to faster to process default nearest neighbor

```
SELECT ST_CreateOverview('mydata.mytable'::regclass, 'rast', 2);
```

Veja também

ST_Retile, AddOverviewConstraints, AddRasterConstraints, Section 9.2.2

10.3 Construtores Raster

10.3.1 ST_AddBand

ST_AddBand — Retorna um raster com nova banda(s) do tipo dado adicionado com o valor inicial com a localização do índice. Se nenhum índice for especificado, a banda é adicionada ao final.

Synopsis

(1) raster **ST_AddBand**(raster rast, addbandarg[] addbandargset);
(2) raster **ST_AddBand**(raster rast, integer index, text pixeltype, double precision initialvalue=0, double precision nodataval=NULL);
(3) raster **ST_AddBand**(raster rast, text pixeltype, double precision initialvalue=0, double precision nodataval=NULL);
(4) raster **ST_AddBand**(raster torast, raster fromrast, integer fromband=1, integer torastindex=at_end);
(5) raster **ST_AddBand**(raster torast, raster[] fromrasts, integer fromband=1, integer torastindex=at_end);
(6) raster **ST_AddBand**(raster rast, integer index, text outdbfile, integer[] outdbindex, double precision nodataval=NULL);
(7) raster **ST_AddBand**(raster rast, text outdbfile, integer[] outdbindex, integer index=at_end, double precision nodataval=NULL);

Descrição

Retorna um raster com uma nova banda adicionada na posição (índice), do dado tipo, do valor inicial, e do dado valor nodata. Se nenhu índice for especificado, a banda é adicionada ao final. Se nenhum `fromband` for especificado, banda 1 é assumida. O tipo pixel é uma representação de string de um dos tipos de pixel especificados em ST_BandPixelType. Se um índice existente for especificado todas as bandas subsequentes >= aquele índice é incrementado por 1. Se um valor inicial maior que o máximo do tipo pixel for especificado, então ele é estabelecido como o maior valor permitido pelo tipo pixel.

Para a variante que pega um arranjo de addbandarg (Variante 1), um valor de índice addbandarg's específico é relativo ao raster no mesmo tempo que a banda é descrita por aquele addbandarg's está sendo adicionada ao raster. Veja o exemplo abaixo.

Para a variante que pega um arranjo de rasters (Variante 5), se `torast` é NULO então a `fromband` banda de cada raster no arranjo está acumulada dentro de um novo raster.

Para as variantes que pegam `outdbfile` (Variantes 6 e 7), o valor deve incluir o caminho completo para o arquivo raster. O arquivo deve ser acessível também pata o processo do servidor postgres.

Melhorias: 2.1.0 suporte para addbandarg adicionado.

Melhorias: 2.1.0 suporte para novas bandas out-db adicionado.

Exemplos: Nova banda única

```
-- Add another band of type 8 bit unsigned integer with pixels initialized to 200
UPDATE dummy_rast
    SET rast = ST_AddBand(rast,'8BUI'::text,200)
WHERE rid = 1;
```

```
-- Create an empty raster 100x100 units, with upper left  right at 0, add 2 bands (band 1  ↩
    is 0/1 boolean bit switch, band2 allows values 0-15)
-- uses addbandargs
INSERT INTO dummy_rast(rid,rast)
    VALUES(10, ST_AddBand(ST_MakeEmptyRaster(100, 100, 0, 0, 1, -1, 0, 0, 0),
    ARRAY[
        ROW(1, '1BB'::text, 0, NULL),
```

```
        ROW(2, '4BUI'::text, 0, NULL)
            ]::addbandarg[]
     )
    );

-- output meta data of raster bands to verify all is right --
SELECT  (bmd).*
FROM (SELECT ST_BandMetaData(rast,generate_series(1,2)) As bmd
    FROM dummy_rast WHERE rid = 10) AS foo;
 --result --
 pixeltype | nodatavalue | isoutdb | path
-----------+-------------+---------+------
 1BB       |             | f       |
 4BUI      |             | f       |


-- output meta data of raster -
SELECT  (rmd).width, (rmd).height, (rmd).numbands
FROM (SELECT ST_MetaData(rast) As rmd
    FROM dummy_rast WHERE rid = 10) AS foo;
-- result --
 upperleftx | upperlefty | width | height | scalex | scaley | skewx | skewy | srid |  ↩
     numbands
------------+------------+-------+--------+------------+------------+-------+-------+------+--------

          0 |          0 |   100 |    100 |      1 |     -1 |     0 |     0 |   0 |         ↩
              2
```

Exemplos: Várias bandas novas

```
SELECT
    *
FROM ST_BandMetadata(
    ST_AddBand(
        ST_MakeEmptyRaster(10, 10, 0, 0, 1, -1, 0, 0, 0),
        ARRAY[
            ROW(NULL, '8BUI', 255, 0),
            ROW(NULL, '16BUI', 1, 2),
            ROW(2, '32BUI', 100, 12),
            ROW(2, '32BF', 3.14, -1)
        ]::addbandarg[]
    ),
    ARRAY[]::integer[]
);

 bandnum | pixeltype | nodatavalue | isoutdb | path
---------+-----------+-------------+---------+------
       1 | 8BUI      |           0 | f       |
       2 | 32BF      |          -1 | f       |
       3 | 32BUI     |          12 | f       |
       4 | 16BUI     |           2 | f       |
```

```
-- Aggregate the 1st band of a table of like rasters into a single raster
-- with as many bands as there are test_types and as many rows (new rasters) as there are  ↩
    mice
-- NOTE: The ORDER BY test_type is only supported in PostgreSQL 9.0+
-- for 8.4 and below it usually works to order your data in a subselect (but not guaranteed ↩
    )
-- The resulting raster will have a band for each test_type alphabetical by test_type
-- For mouse lovers: No mice were harmed in this exercise
```

```
SELECT
    mouse,
    ST_AddBand(NULL, array_agg(rast ORDER BY test_type), 1) As rast
FROM mice_studies
GROUP BY mouse;
```

Exemplos: Nova banda out-db

```
SELECT
    *
FROM ST_BandMetadata(
    ST_AddBand(
        ST_MakeEmptyRaster(10, 10, 0, 0, 1, -1, 0, 0, 0),
        '/home/raster/mytestraster.tif'::text, NULL::int[]
    ),
    ARRAY[]::integer[]
);

 bandnum | pixeltype | nodatavalue | isoutdb | path
---------+-----------+-------------+---------+------
       1 | 8BUI      |             | t       | /home/raster/mytestraster.tif
       2 | 8BUI      |             | t       | /home/raster/mytestraster.tif
       3 | 8BUI      |             | t       | /home/raster/mytestraster.tif
```

Veja também

ST_BandMetaData, ST_BandPixelType, ST_MakeEmptyRaster, ST_MetaData, ST_NumBands, ST_Reclass

10.3.2 ST_AsRaster

ST_AsRaster — Converte uma geometria PostGIS para um raster PostGIS.

Synopsis

raster **ST_AsRaster**(geometry geom, raster ref, text pixeltype, double precision value=1, double precision nodataval=0, boolean touched=false);
raster **ST_AsRaster**(geometry geom, raster ref, text[] pixeltype=ARRAY['8BUI'], double precision[] value=ARRAY[1], double precision[] nodataval=ARRAY[0], boolean touched=false);
raster **ST_AsRaster**(geometry geom, double precision scalex, double precision scaley, double precision gridx, double precision gridy, text pixeltype, double precision value=1, double precision nodataval=0, double precision skewx=0, double precision skewy=0, boolean touched=false);
raster **ST_AsRaster**(geometry geom, double precision scalex, double precision scaley, double precision gridx=NULL, double precision gridy=NULL, text[] pixeltype=ARRAY['8BUI'], double precision[] value=ARRAY[1], double precision[] nodataval=ARRAY[0], double precision skewx=0, double precision skewy=0, boolean touched=false);
raster **ST_AsRaster**(geometry geom, double precision scalex, double precision scaley, text pixeltype, double precision value=1, double precision nodataval=0, double precision upperleftx=NULL, double precision upperlefty=NULL, double precision skewx=0, double precision skewy=0, boolean touched=false);
raster **ST_AsRaster**(geometry geom, double precision scalex, double precision scaley, text[] pixeltype, double precision[] value=ARRAY[1], double precision[] nodataval=ARRAY[0], double precision upperleftx=NULL, double precision upperlefty=NULL, double precision skewx=0, double precision skewy=0, boolean touched=false);
raster **ST_AsRaster**(geometry geom, integer width, integer height, double precision gridx, double precision gridy, text pixeltype, double precision value=1, double precision nodataval=0, double precision skewx=0, double precision skewy=0, boolean touched=false);
raster **ST_AsRaster**(geometry geom, integer width, integer height, double precision gridx=NULL, double precision gridy=NULL,

text[] pixeltype=ARRAY['8BUI'], double precision[] value=ARRAY[1], double precision[] nodataval=ARRAY[0], double precision skewx=0, double precision skewy=0, boolean touched=false);
raster **ST_AsRaster**(geometry geom, integer width, integer height, text pixeltype, double precision value=1, double precision nodataval=0, double precision upperleftx=NULL, double precision upperlefty=NULL, double precision skewx=0, double precision skewy=0, boolean touched=false);
raster **ST_AsRaster**(geometry geom, integer width, integer height, text[] pixeltype, double precision[] value=ARRAY[1], double precision[] nodataval=ARRAY[0], double precision upperleftx=NULL, double precision upperlefty=NULL, double precision skewx=0, double precision skewy=0, boolean touched=false);

Descrição

Converte uma geometria PostGIS para um raster PostGIS. As diversas variantes oferecem três grupos de possibilidades para configurar o alinhamento e o tamanho do pixel do raster resultante.

O primeiro grupo, composto pelas duas primeiras variantes, produz um raster tendo o mesmo alinhamento (`scalex`, `scaley`, `gridx` e `gridy`), tipo pixel e valor nodata como foi fornecido pelo raster referência. Você geralmente passa este raster referência unindo a tabela que contém a geometria com a que contém o raster referência.

O segundo grupo, composto por quatro variantes, permite que você fixe dimensões do raster fornecendo os parâmetros do tamanho de um pixel (`scalex` & `scaley` e `skewx` & `skewy`). O `width` & `height` do raster resultante será ajustado para caber na extensão da geometria. Na maioria dos casos, você deve cast integer `scalex` & `scaley` argumentos para dobrar a precisão para que o PostgreSQL escolha a variante correta.

O terceiro grupo, composto por quatro variantes, permite que você conserte dimensões do raster fornecendo elas (`width` & `height`). Os parâmetros do tamanho do pixel (`scalex` & `scaley` and `skewx` & `skewy`) do raster resultante será ajustado para caber na extensão da geometria.

As duas primeiras variantes de cada um destes dois últimos grupos permite que você especifique o alinhamento com um canto aleatório da rede de alinhamento (`gridx` & `gridy`) e as duas últimas variantes pegam o canto esquerdo mais alto (`upperleftx` & `upperlefty`).

Cada grupo de variantes permite a produção de uma ou várias bandas raster. Para produzir várias, você deve fornecer um arranjo de tipos pixel (`pixeltype[]`), um arranjo de valores iniciais (`value`) e um de valores nodata (`nodataval`). Se não fornecidos pixeltyped torna-se 8BUI, valores para 1 e nodataval para 0.

O raster de saída terá a mesma referência espacial que a geometria fonte. A única exceção é para variantes com raster referência. Neste caso, o raster resultante terá o mesmo SRID do raster referência.

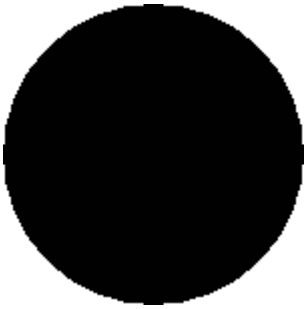
O parâmetro opcional `touched` é falso e mapeia a opção rasterização GDAL ALL_TOUCHED, a qual determina se pixeis tocados por linhas ou polígonos serão queimados. Não apenas aqueles no caminho de renderização de linha, ou aqueles cujo ponto central está dentro do polígono.

Isto é particularmente útil para renderizar jpegs e pngs de geometrias diretamente do banco de dados quando usando em conjunto com ST_AsPNG e outra família ST_AsGDALRaster de funções.

Disponibilidade: 2.0.0 - requer GDAL >= 1.6.0.

Note
Ainda não é capaz de renderizar geometrias complexas como: curvas, TINS, e superfícies poliédricas, mas deveria ser, já que GDAL consegue.

Exemplos: Gera geometrias como arquivos PNG



círculo preto

```
-- this will output a black circle taking up 150 x 150 pixels --
SELECT ST_AsPNG(ST_AsRaster(ST_Buffer(ST_Point(1,5),10),150, 150));
```

exemplo de buffer renderizado só com o PostGIS

```
-- the bands map to RGB bands - the value (118,154,118) - teal  --
SELECT ST_AsPNG(
    ST_AsRaster(
        ST_Buffer(
            ST_GeomFromText('LINESTRING(50 50,150 150,150 50)'), 10,'join=bevel'),
            200,200,ARRAY['8BUI', '8BUI', '8BUI'], ARRAY[118,154,118], ARRAY[0,0,0]));
```

Veja também

ST_BandPixelType, ST_Buffer, ST_GDALDrivers, ST_AsGDALRaster, ST_AsPNG, ST_AsJPEG, ST_SRID

10.3.3 ST_Band

ST_Band — Retorna uma ou mais bandas de um raster existente como um novo raster. Útil para a construção de novos rasters a partir de rasters existentes.

Synopsis

raster **ST_Band**(raster rast, integer[] nbands = ARRAY[1]);
raster **ST_Band**(raster rast, integer nband);
raster **ST_Band**(raster rast, text nbands, character delimiter=,);

Descrição

Returns one or more bands of an existing raster as a new raster. Useful for building new rasters from existing rasters or export of only selected bands of a raster or rearranging the order of bands in a raster. If no band is specified or any of specified bands does not exist in the raster, then all bands are returned. Used as a helper function in various functions such as for deleting a band.

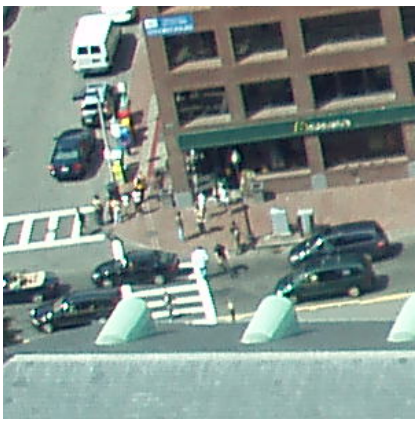
Warning
Para as `nbands` como variantes de textos de função, o delimitador padrão é `,` que significa que você pode pedir por `'1,2,3'` e se quiser usar um delimitador diferente você poderia fazer `ST_Band(rast, '1@2@3', '@')`. Para pedir por várias bandas, sugerimos que use a forma de arranjo desta função ex.: `ST_Band(rast, '{1,2,3}'::int[])`; já que a forma de lista de banda `text` pode ser removida em versões futuras do PostGIS.

Disponibilidade: 2.0.0

Exemplos

```
-- Make 2 new rasters: 1 containing band 1 of dummy, second containing band 2 of dummy and  ↩
    then reclassified as a 2BUI
SELECT ST_NumBands(rast1) As numb1, ST_BandPixelType(rast1) As pix1,
 ST_NumBands(rast2) As numb2,  ST_BandPixelType(rast2) As pix2
FROM (
    SELECT ST_Band(rast) As rast1, ST_Reclass(ST_Band(rast,3), '100-200):1, [200-254:2', '2 ↩
        BUI') As rast2
        FROM dummy_rast
        WHERE rid = 2) As foo;

 numb1 | pix1 | numb2 | pix2
-------+------+-------+------
     1 | 8BUI |     1 | 2BUI
```

```
-- Return bands 2 and 3. Using array cast syntax
SELECT ST_NumBands(ST_Band(rast, '{2,3}'::int[])) As num_bands
    FROM dummy_rast WHERE rid=2;

num_bands
----------
2

-- Return bands 2 and 3. Use array to define bands
SELECT ST_NumBands(ST_Band(rast, ARRAY[2,3])) As num_bands
    FROM dummy_rast
WHERE rid=2;
```

		
original (column rast)	*dupe_band*	*sing_band*

```
--Make a new raster with 2nd band of original and 1st band repeated twice,
and another with just the third band
SELECT rast, ST_Band(rast, ARRAY[2,1,1]) As dupe_band,
    ST_Band(rast, 3) As sing_band
FROM samples.than_chunked
WHERE rid=35;
```

Veja também

ST_AddBand, ST_NumBands, ST_Reclass, Chapter 10

10.3.4 ST_MakeEmptyCoverage

ST_MakeEmptyCoverage — Cover georeferenced area with a grid of empty raster tiles.

Synopsis

raster **ST_MakeEmptyCoverage**(integer tilewidth, integer tileheight, integer width, integer height, double precision upperleftx, double precision upperlefty, double precision scalex, double precision scaley, double precision skewx, double precision skewy, integer srid=unknown);

Descrição

Create a set of raster tiles with ST_MakeEmptyRaster. Grid dimension is `width` & `height`. Tile dimension is `tilewidth` & `tileheight`. The covered georeferenced area is from upper left corner (`upperleftx`, `upperlefty`) to lower right corner (`upperleftx` + `width` * `scalex`, `upperlefty` + `height` * `scaley`).

Note
Note that scaley is generally negative for rasters and scalex is generally positive. So lower right corner will have a lower y value and higher x value than the upper left corner.

Availability: 2.4.0

Exemplos básicos

Create 16 tiles in a 4x4 grid to cover the WGS84 area from upper left corner (22, 77) to lower right corner (55, 33).

```
SELECT (ST_MetaData(tile)).* FROM ST_MakeEmptyCoverage(1, 1, 4, 4, 22, 33, (55 - 22)/(4):: ↩
    float, (33 - 77)/(4)::float, 0., 0., 4326) tile;

 upperleftx | upperlefty | width | height | scalex | scaley | skewx | skewy | srid |  ↩
     numbands
------------------------------------------------------------------------------------------
         22 |         33 |     1 |      1 |   8.25 |    -11 |     0 |     0 | 4326 |  ↩
                  0
      30.25 |         33 |     1 |      1 |   8.25 |    -11 |     0 |     0 | 4326 |  ↩
                  0
       38.5 |         33 |     1 |      1 |   8.25 |    -11 |     0 |     0 | 4326 |  ↩
                  0
      46.75 |         33 |     1 |      1 |   8.25 |    -11 |     0 |     0 | 4326 |  ↩
                  0
         22 |         22 |     1 |      1 |   8.25 |    -11 |     0 |     0 | 4326 |  ↩
                  0
      30.25 |         22 |     1 |      1 |   8.25 |    -11 |     0 |     0 | 4326 |  ↩
                  0
       38.5 |         22 |     1 |      1 |   8.25 |    -11 |     0 |     0 | 4326 |  ↩
                  0
      46.75 |         22 |     1 |      1 |   8.25 |    -11 |     0 |     0 | 4326 |  ↩
                  0
         22 |         11 |     1 |      1 |   8.25 |    -11 |     0 |     0 | 4326 |  ↩
                  0
      30.25 |         11 |     1 |      1 |   8.25 |    -11 |     0 |     0 | 4326 |  ↩
                  0
       38.5 |         11 |     1 |      1 |   8.25 |    -11 |     0 |     0 | 4326 |  ↩
                  0
      46.75 |         11 |     1 |      1 |   8.25 |    -11 |     0 |     0 | 4326 |  ↩
                  0
         22 |          0 |     1 |      1 |   8.25 |    -11 |     0 |     0 | 4326 |  ↩
                  0
      30.25 |          0 |     1 |      1 |   8.25 |    -11 |     0 |     0 | 4326 |  ↩
                  0
       38.5 |          0 |     1 |      1 |   8.25 |    -11 |     0 |     0 | 4326 |  ↩
                  0
      46.75 |          0 |     1 |      1 |   8.25 |    -11 |     0 |     0 | 4326 |  ↩
                  0
```

Veja também

ST_MakeEmptyRaster

10.3.5 ST_MakeEmptyRaster

ST_MakeEmptyRaster — Retorna um raster vazio (sem bandas) das dimensões dadas (width & height), o X e Y do superior esquerdo, tamanho de pixel e rotação (scalex, scaley, skewx & skewy) e sistema de referência (srid). Se um raster passar, retorna um novo raster com o mesmo tamanho, alinhamento e SRID. Se o srid é deixado de fora, a referência espacial se torna desconhecida (0).

Synopsis

raster **ST_MakeEmptyRaster**(raster rast);
raster **ST_MakeEmptyRaster**(integer width, integer height, float8 upperleftx, float8 upperlefty, float8 scalex, float8 scaley,

float8 skewx, float8 skewy, integer srid=unknown);
raster **ST_MakeEmptyRaster**(integer width, integer height, float8 upperleftx, float8 upperlefty, float8 pixelsize);

Descrição

Retorna um raster vazio (sem bandas) das dimensões dadas (width & height) e georeferenciado nas coordenadas espaciais (ou mundo) com o X esquerdo superior (upperleftx), Y superior esquerdo (upperlefty), tamanho de pixel e rotação (scalex, scaley, skewx & skewy) e sistema de referência (srid).

A última versão usa um único parâmetro para especificar o tamanho do pixel (pixelsize). scalex é estabelecida neste argumento e scaley é estabelecida no valor negativo deste argumento. skewx e skewy são 0.

Se um raster existente passar, ele retorna um novo raster com as mesmas configurações de meta dados (sem as bandas).

Se nenhum srid é especificado, é 0. Depois que você criou um raster vazio talvez queira adicionar bandas a ele e editá-lo. Recorra a ST_AddBand para definir bandas e ST_SetValue para valores iniciais de pixel.

Exemplos

```
INSERT INTO dummy_rast(rid,rast)
VALUES(3, ST_MakeEmptyRaster( 100, 100, 0.0005, 0.0005, 1, 1, 0, 0, 4326) );

--use an existing raster as template for new raster
INSERT INTO dummy_rast(rid,rast)
SELECT 4, ST_MakeEmptyRaster(rast)
FROM dummy_rast WHERE rid = 3;

-- output meta data of rasters we just added
SELECT rid, (md).*
FROM (SELECT rid, ST_MetaData(rast) As md
    FROM dummy_rast
    WHERE rid IN(3,4)) As foo;

-- output --
 rid | upperleftx | upperlefty | width | height | scalex | scaley | skewx | skewy | srid |  ↩
     numbands
-----+------------+------------+-------+--------+------------+------------+-------+-------+------+--

   3 |     0.0005 |     0.0005 |   100 |    100 |          1 |          1 |    0  |     0 | ↩
       4326 |        0
   4 |     0.0005 |     0.0005 |   100 |    100 |          1 |          1 |    0  |     0 | ↩
       4326 |        0
```

Veja também

ST_AddBand, ST_MetaData, ST_ScaleX, ST_ScaleY, ST_SetValue, ST_SkewX, , ST_SkewY

10.3.6 ST_Tile

ST_Tile — Retorna um conjunto de rasters resultante de uma divisão do raster de entrada baseado nas dimensões desejadas nos rasters de saída.

Synopsis

setof raster **ST_Tile**(raster rast, int[] nband, integer width, integer height, boolean padwithnodata=FALSE, double precision nodataval=NULL);

setof raster **ST_Tile**(raster rast, integer nband, integer width, integer height, boolean padwithnodata=FALSE, double precision nodataval=NULL);
setof raster **ST_Tile**(raster rast, integer width, integer height, boolean padwithnodata=FALSE, double precision nodataval=NULL);

Descrição

Retorna um conjunto de rasters resultante de uma divisão do raster de entrada baseado nas dimensões desejadas nos rasters de saída.

Se `padwithnodata` = FALSO, tiles limite no lado direito e inferior do raster podem ter dimensões diferentes do resto das tiles. Se `padwithnodata` = VERDADEIRO, todas as tiles terão a mesma dimensão com a possibilidade das tiles limites serem preenchidas com valores NODATA. Se a banda(s) não possui valor(es) NODATA especificado, pode ser especificado por `nodataval`.

Note
Se uma banda especificada do raster de entrada estiver out-of-db, a banda correspondente nos rasters de saída também estará.

Disponibilidade: 2.1.0

Exemplos

```
WITH foo AS (
    SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0), 1, '8BUI', ↩
        1, 0), 2, '8BUI', 10, 0) AS rast UNION ALL
    SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 3, 0, 1, -1, 0, 0, 0), 1, '8BUI', ↩
        2, 0), 2, '8BUI', 20, 0) AS rast UNION ALL
    SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 6, 0, 1, -1, 0, 0, 0), 1, '8BUI', ↩
        3, 0), 2, '8BUI', 30, 0) AS rast UNION ALL

    SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, -3, 1, -1, 0, 0, 0), 1, '8BUI ↩
        ', 4, 0), 2, '8BUI', 40, 0) AS rast UNION ALL
    SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 3, -3, 1, -1, 0, 0, 0), 1, '8BUI ↩
        ', 5, 0), 2, '8BUI', 50, 0) AS rast UNION ALL
    SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 6, -3, 1, -1, 0, 0, 0), 1, '8BUI ↩
        ', 6, 0), 2, '8BUI', 60, 0) AS rast UNION ALL

    SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, -6, 1, -1, 0, 0, 0), 1, '8BUI ↩
        ', 7, 0), 2, '8BUI', 70, 0) AS rast UNION ALL
    SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 3, -6, 1, -1, 0, 0, 0), 1, '8BUI ↩
        ', 8, 0), 2, '8BUI', 80, 0) AS rast UNION ALL
    SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 6, -6, 1, -1, 0, 0, 0), 1, '8BUI ↩
        ', 9, 0), 2, '8BUI', 90, 0) AS rast
), bar AS (
    SELECT ST_Union(rast) AS rast FROM foo
), baz AS (
    SELECT ST_Tile(rast, 3, 3, TRUE) AS rast FROM bar
)
SELECT
    ST_DumpValues(rast)
FROM baz;

              st_dumpvalues
---------------------------------------
 (1,"{{1,1,1},{1,1,1},{1,1,1}}")
 (2,"{{10,10,10},{10,10,10},{10,10,10}}")
 (1,"{{2,2,2},{2,2,2},{2,2,2}}")
```

```
 (2,"{{20,20,20},{20,20,20},{20,20,20}}")
 (1,"{{3,3,3},{3,3,3},{3,3,3}}")
 (2,"{{30,30,30},{30,30,30},{30,30,30}}")
 (1,"{{4,4,4},{4,4,4},{4,4,4}}")
 (2,"{{40,40,40},{40,40,40},{40,40,40}}")
 (1,"{{5,5,5},{5,5,5},{5,5,5}}")
 (2,"{{50,50,50},{50,50,50},{50,50,50}}")
 (1,"{{6,6,6},{6,6,6},{6,6,6}}")
 (2,"{{60,60,60},{60,60,60},{60,60,60}}")
 (1,"{{7,7,7},{7,7,7},{7,7,7}}")
 (2,"{{70,70,70},{70,70,70},{70,70,70}}")
 (1,"{{8,8,8},{8,8,8},{8,8,8}}")
 (2,"{{80,80,80},{80,80,80},{80,80,80}}")
 (1,"{{9,9,9},{9,9,9},{9,9,9}}")
 (2,"{{90,90,90},{90,90,90},{90,90,90}}")
(18 rows)
```

```
WITH foo AS (
    SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0), 1, '8BUI', ↩
         1, 0), 2, '8BUI', 10, 0) AS rast UNION ALL
    SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 3, 0, 1, -1, 0, 0, 0), 1, '8BUI', ↩
         2, 0), 2, '8BUI', 20, 0) AS rast UNION ALL
    SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 6, 0, 1, -1, 0, 0, 0), 1, '8BUI', ↩
         3, 0), 2, '8BUI', 30, 0) AS rast UNION ALL

    SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, -3, 1, -1, 0, 0, 0), 1, '8BUI ↩
        ', 4, 0), 2, '8BUI', 40, 0) AS rast UNION ALL
    SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 3, -3, 1, -1, 0, 0, 0), 1, '8BUI ↩
        ', 5, 0), 2, '8BUI', 50, 0) AS rast UNION ALL
    SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 6, -3, 1, -1, 0, 0, 0), 1, '8BUI ↩
        ', 6, 0), 2, '8BUI', 60, 0) AS rast UNION ALL

    SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, -6, 1, -1, 0, 0, 0), 1, '8BUI ↩
        ', 7, 0), 2, '8BUI', 70, 0) AS rast UNION ALL
    SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 3, -6, 1, -1, 0, 0, 0), 1, '8BUI ↩
        ', 8, 0), 2, '8BUI', 80, 0) AS rast UNION ALL
    SELECT ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 6, -6, 1, -1, 0, 0, 0), 1, '8BUI ↩
        ', 9, 0), 2, '8BUI', 90, 0) AS rast
), bar AS (
    SELECT ST_Union(rast) AS rast FROM foo
), baz AS (
    SELECT ST_Tile(rast, 3, 3, 2) AS rast FROM bar
)
SELECT
    ST_DumpValues(rast)
FROM baz;

              st_dumpvalues
-----------------------------------------
 (1,"{{10,10,10},{10,10,10},{10,10,10}}")
 (1,"{{20,20,20},{20,20,20},{20,20,20}}")
 (1,"{{30,30,30},{30,30,30},{30,30,30}}")
 (1,"{{40,40,40},{40,40,40},{40,40,40}}")
 (1,"{{50,50,50},{50,50,50},{50,50,50}}")
 (1,"{{60,60,60},{60,60,60},{60,60,60}}")
 (1,"{{70,70,70},{70,70,70},{70,70,70}}")
 (1,"{{80,80,80},{80,80,80},{80,80,80}}")
 (1,"{{90,90,90},{90,90,90},{90,90,90}}")
(9 rows)
```

Veja também

ST_Union, ST_Retile

10.3.7 ST_Retile

ST_Retile — Retorna um conjunto de tiles configuradas de uma cobertura raster aleatória.

Synopsis

SETOF raster **ST_Retile**(regclass tab, name col, geometry ext, float8 sfx, float8 sfy, int tw, int th, text algo='NearestNeighbor');

Descrição

Retorna um conjunto de tiles tendo a escala especificada (`sfx`, `sfy`) e tamanho máximo (`tw`, `th`) e cobrindo a extensão especificada (`ext`) com os dados vindos da cobertura raster especificada (`tab`, `col`).

As opções de algorítimo são: 'NearestNeighbor', 'Bilinear', 'Cubic', 'CubicSpline', e 'Lanczos'. Recorra a: GDAL Warp resampling methods para mais detalhes.

Disponibilidade: 2.2.0

Veja também

ST_CreateOverview

10.3.8 ST_FromGDALRaster

ST_FromGDALRaster — Retorna um raster de um arquivo raster GDAL suportado.

Synopsis

raster **ST_FromGDALRaster**(bytea gdaldata, integer srid=NULL);

Descrição

Retorna um raster de uma arquivo raster GDAL suportado. `gdaldata` é do tipo bytea e deve ser o conteúdo do arquivo raster GDAL.

Se `srid` for NULO, a função tentará designar automaticamente o SRID do raster GDAL. Se `srid` for fornecido, o valor fornecido irá exceder qualquer SRID designado automaticamente.

Disponibilidade: 2.1.0

Exemplos

```
WITH foo AS (
    SELECT ST_AsPNG(ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 0.1,  ↩
        -0.1, 0, 0, 4326), 1, '8BUI', 1, 0), 2, '8BUI', 2, 0), 3, '8BUI', 3, 0)) AS png
),
bar AS (
    SELECT 1 AS rid, ST_FromGDALRaster(png) AS rast FROM foo
    UNION ALL
    SELECT 2 AS rid, ST_FromGDALRaster(png, 3310) AS rast FROM foo
```

```
)
SELECT
    rid,
    ST_Metadata(rast) AS metadata,
    ST_SummaryStats(rast, 1) AS stats1,
    ST_SummaryStats(rast, 2) AS stats2,
    ST_SummaryStats(rast, 3) AS stats3
FROM bar
ORDER BY rid;

 rid |          metadata           |    stats1     |    stats2     |     stats3
-----+-----------------------------+---------------+---------------+----------------
   1 | (0,0,2,2,1,-1,0,0,0,3)      | (4,4,1,0,1,1) | (4,8,2,0,2,2) | (4,12,3,0,3,3)
   2 | (0,0,2,2,1,-1,0,0,3310,3)   | (4,4,1,0,1,1) | (4,8,2,0,2,2) | (4,12,3,0,3,3)
(2 rows)
```

Veja também

ST_AsGDALRaster

10.4 Assessores Raster

10.4.1 ST_GeoReference

ST_GeoReference — Retorna os metadados georreferenciados no formato GDAL ou ESRI como é comumente visto em um arquivo mundo. O padrão é GDAL.

Synopsis

text **ST_GeoReference**(raster rast, text format=GDAL);

Descrição

Retorna os metadados georreferenciados, incluindo transporte, no formato GDAL ou ESRI como visto no `srid`. O padrão é GDAL se nenhum tipo for especificado. O tipo é string 'GDAL' ou 'ESRI'.

A diferença entre representações de formatos é a seguinte:

`GDAL`:

```
scalex
skewy
skewx
scaley
upperleftx
upperlefty
```

`ESRI`:

```
scalex
skewy
skewx
scaley
upperleftx + scalex*0.5
upperlefty + scaley*0.5
```

Exemplos

```
SELECT ST_GeoReference(rast, 'ESRI') As esri_ref, ST_GeoReference(rast, 'GDAL') As gdal_ref
 FROM dummy_rast WHERE rid=1;

   esri_ref    |   gdal_ref
---------------+--------------
 2.0000000000 | 2.0000000000
 0.0000000000 : 0.0000000000
 0.0000000000 : 0.0000000000
 3.0000000000 : 3.0000000000
 1.5000000000 : 0.5000000000
 2.0000000000 : 0.5000000000
```

Veja também

ST_SetGeoReference, ST_ScaleX, ST_ScaleY

10.4.2 ST_Height

ST_Height — Retorna a altura do raster em pixeis.

Synopsis

integer **ST_Height**(raster rast);

Descrição

Retorna a altura do raster.

Exemplos

```
SELECT rid, ST_Height(rast) As rastheight
FROM dummy_rast;

 rid | rastheight
-----+------------
   1 |         20
   2 |          5
```

Veja também

ST_Width

10.4.3 ST_IsEmpty

ST_IsEmpty — Retorna verdadeiro se o raster estiver vazio (largura = 0 e altura = 0). Senão, retorna falso.

Synopsis

boolean **ST_IsEmpty**(raster rast);

Descrição

Retorna verdadeiro se o raster estiver vazio (largura = 0 e altura = 0). Senão, retorna falso.

Disponibilidade: 2.0.0

Exemplos

```
SELECT ST_IsEmpty(ST_MakeEmptyRaster(100, 100, 0, 0, 0, 0, 0, 0))
st_isempty |
-----------+
f          |


SELECT ST_IsEmpty(ST_MakeEmptyRaster(0, 0, 0, 0, 0, 0, 0, 0))
st_isempty |
-----------+
t          |
```

Veja também

ST_HasNoBand

10.4.4 ST_MemSize

ST_MemSize — Retorna a quantidade de espaço (em bytes) que o raster pega.

Synopsis

integer **ST_MemSize**(raster rast);

Descrição

Retorna a quantidade de espaço (em bytes) que o raster pega.

Isso é um ótimo elogio para o PostgreSQL construído nas funções pg_column_size, pg_size_pretty, pg_relation_size, pg_total_relation_s

Note
pg_relation_size que fornece o tamanho em bytes de uma tabela, talvez retorne com o tamanho em byte menor que ST_MemSize. Isso acontece porque o pg_relation_size não adiciona contribuição de tabela toasted e grandes geometrias são guardadas em tabelas TOAST. g_column_size pode retornar menor porque retorna o tamanho comprimido. pg_total_relation_size - inclui, a tabela, as tabelas toasted, e os índices.

Disponibilidade: 2.2.0

Exemplos

```
SELECT ST_MemSize(ST_AsRaster(ST_Buffer(ST_Point(1,5),10,1000),150, 150, '8BUI')) As  ↩
    rast_mem;

        rast_mem
        --------
        22568
```

Veja também

10.4.5 ST_MetaData

ST_MetaData — Retorna metadados básicos sobre um objeto raster como um tanho pixel, rotação (skew), esquerda superior, inferior etc.

Synopsis

record **ST_MetaData**(raster rast);

Descrição

Retorna metadados básicos sobre um objeto raster como um tanho pixel, rotação (skew), esquerda superior, inferior etc. Colunas retornadas: upperleftx | upperlefty | width | height | scalex | scaley | skewx | skewy | srid | numbands

Exemplos

```
SELECT rid, (foo.md).*
 FROM (SELECT rid, ST_MetaData(rast) As md
FROM dummy_rast) As foo;

 rid | upperleftx | upperlefty | width | height | scalex | scaley | skewx | skewy | srid |  ↩
     numbands
 ----+------------+------------+-------+--------+--------+-----------+-------+-------+------+-------

   1 |        0.5 |        0.5 |    10 |     20 |      2 |         3 |     0 |     0 |    0 |  ↩
             0
   2 | 3427927.75 |    5793244 |     5 |      5 |   0.05 |     -0.05 |     0 |     0 |    0 |  ↩
             3
```

Veja também

ST_BandMetaData, ST_NumBands

10.4.6 ST_NumBands

ST_NumBands — Retorna o número de bandas no objeto raster.

Synopsis

integer **ST_NumBands**(raster rast);

Descrição

Retorna o número de bandas no objeto raster.

Exemplos

```
SELECT rid, ST_NumBands(rast) As numbands
FROM dummy_rast;

rid | numbands
----+----------
  1 |        0
  2 |        3
```

Veja também

ST_Value

10.4.7 ST_PixelHeight

ST_PixelHeight — Retorna a altura do pixel em unidades geométricas do sistema de referência espacial.

Synopsis

double precision **ST_PixelHeight**(raster rast);

Descrição

Retorna a altura do pixel em unidades geométricas do sistema de referência espacial. No caso comum onde não existem desvios, a altura do pixel é somente a escala de proporção entre coordenadas geométricas e pixeis raster.

Recorra a ST_PixelWidth para uma visualização diagramática da relação.

Exemplos: Rasters sem desvio

```
SELECT ST_Height(rast) As rastheight, ST_PixelHeight(rast) As pixheight,
 ST_ScaleX(rast) As scalex, ST_ScaleY(rast) As scaley, ST_SkewX(rast) As skewx,
        ST_SkewY(rast) As skewy
FROM dummy_rast;

 rastheight | pixheight | scalex | scaley | skewx | skewy
------------+-----------+--------+--------+-------+----------
         20 |         3 |      2 |      3 |     0 |        0
          5 |      0.05 |   0.05 |  -0.05 |     0 |        0
```

Exemplos: Rasters com desvio diferente de 0

```
SELECT ST_Height(rast) As rastheight, ST_PixelHeight(rast) As pixheight,
 ST_ScaleX(rast) As scalex, ST_ScaleY(rast) As scaley, ST_SkewX(rast) As skewx,
        ST_SkewY(rast) As skewy
FROM (SELECT ST_SetSKew(rast,0.5,0.5) As rast
        FROM dummy_rast) As skewed;

rastheight |     pixheight     | scalex | scaley | skewx | skewy
-----------+-------------------+--------+--------+-------+----------
        20 |  3.04138126514911 |      2 |      3 |   0.5 |      0.5
         5 | 0.502493781056044 |   0.05 |  -0.05 |   0.5 |      0.5
```

Veja também

ST_PixelWidth, ST_ScaleX, ST_ScaleY, ST_SkewX, ST_SkewY

10.4.8 ST_PixelWidth

ST_PixelWidth — Retorna a largura do pixel em unidades geométricas do sistema de referência espacial.

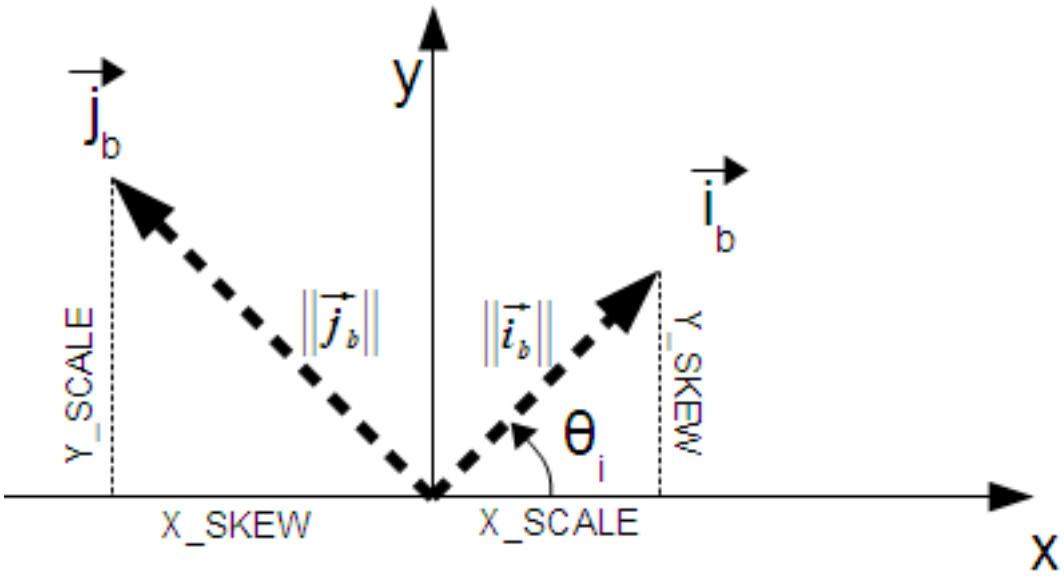
Synopsis

double precision **ST_PixelWidth**(raster rast);

Descrição

Retorna a largura do pixel em unidades geométricas do sistema de referência espacial. No caso comum onde não existem desvios, a largura do pixel é somente a escala de proporção entre coordenadas geométricas e pixeis raster.

O diagrama a seguir demonstra a relação:



Largura do Pixel: tamanho do pixel na direção i
Altura do Pixel: tamanho do pixel na direção j

Exemplos: Rasters sem desvio

```
SELECT ST_Width(rast) As rastwidth, ST_PixelWidth(rast) As pixwidth,
    ST_ScaleX(rast) As scalex, ST_ScaleY(rast) As scaley, ST_SkewX(rast) As skewx,
    ST_SkewY(rast) As skewy
    FROM dummy_rast;

    rastwidth | pixwidth | scalex | scaley | skewx | skewy
    ----------+----------+--------+--------+-------+----------
    10 |         2 |      2 |      3 |     0 |        0
     5 |      0.05 |   0.05 |  -0.05 |     0 |        0
```

Exemplos: Rasters com desvio diferente de 0

```
SELECT ST_Width(rast) As rastwidth, ST_PixelWidth(rast) As pixwidth,
    ST_ScaleX(rast) As scalex, ST_ScaleY(rast) As scaley, ST_SkewX(rast) As skewx,
    ST_SkewY(rast) As skewy
    FROM (SELECT ST_SetSkew(rast,0.5,0.5) As rast
    FROM dummy_rast) As skewed;

    rastwidth |     pixwidth      | scalex | scaley | skewx | skewy
    ----------+-------------------+--------+--------+-------+----------
    10 |  2.06155281280883 |      2 |      3 |   0.5 |      0.5
     5 | 0.502493781056044 |   0.05 |  -0.05 |   0.5 |      0.5
```

Veja também

ST_PixelHeight, ST_ScaleX, ST_ScaleY, ST_SkewX, ST_SkewY

10.4.9 ST_ScaleX

ST_ScaleX — Retorna o componente X da largura do pixel em unidades do sistema de referência coordenadas.

Synopsis

float8 **ST_ScaleX**(raster rast);

Descrição

Retorna o componente X da largura do pixel em unidades do sistema de referência coordenadas. Recorra a World File para mais detalhes.

Alterações: 2.0.0. Nas versões WKTRaster era chamado de ST_PixelSizeX.

Exemplos

```
SELECT rid, ST_ScaleX(rast) As rastpixwidth
FROM dummy_rast;

 rid | rastpixwidth
-----+--------------
   1 |            2
   2 |         0.05
```

Veja também

ST_Width

10.4.10 ST_ScaleY

ST_ScaleY — Retorna o componente Y da altura do pixel em unidades do sistema de referência coordenadas.

Synopsis

float8 **ST_ScaleY**(raster rast);

Descrição

Retorna o componente Y da altura do pixel em unidades do sistema de referência coordenadas. Recorra a World File para mais detalhes.

Alterações: 2.0.0. Nas versões WKTRaster era chamado de ST_PixelSizeY.

Exemplos

```
SELECT rid, ST_ScaleY(rast) As rastpixheight
FROM dummy_rast;

 rid | rastpixheight
-----+---------------
   1 |             3
   2 |         -0.05
```

Veja também

ST_Height

10.4.11 ST_RasterToWorldCoord

ST_RasterToWorldCoord — Retorna o canto superior esquerdo do raster como X e Y geométricos (longitude e latitude) dada a coluna e linha. Coluna e linha começam em 1.

Synopsis

record **ST_RasterToWorldCoord**(raster rast, integer xcolumn, integer yrow);

Descrição

Retorna o canto esquerdo superior como X e Y geométricos (longitude e latitude) dada a coluna e linha. o X e Y estão em unidades geométricas do raster georreferenciado. A numeração da coluna e da linha começa no 1, mas se algum passar como zero, um número negativo ou um maior que a respectiva dimensão do raster, retornará as coordenas fora do raster assumindo que a rede raster é aplicável fora dos limites do raster.

Disponibilidade: 2.1.0

Exemplos

```
-- non-skewed raster
SELECT
    rid,
    (ST_RasterToWorldCoord(rast,1, 1)).*,
    (ST_RasterToWorldCoord(rast,2, 2)).*
FROM dummy_rast

 rid | longitude  | latitude | longitude |  latitude
-----+------------+----------+-----------+------------
   1 |        0.5 |      0.5 |       2.5 |        3.5
   2 | 3427927.75 |  5793244 | 3427927.8 | 5793243.95
```

```
-- skewed raster
SELECT
    rid,
    (ST_RasterToWorldCoord(rast, 1, 1)).*,
    (ST_RasterToWorldCoord(rast, 2, 3)).*
FROM (
    SELECT
        rid,
        ST_SetSkew(rast, 100.5, 0) As rast
    FROM dummy_rast
) As foo

 rid | longitude  | latitude | longitude | latitude
-----+------------+----------+-----------+-----------
   1 |        0.5 |      0.5 |     203.5 |       6.5
   2 | 3427927.75 |  5793244 | 3428128.8 | 5793243.9
```

Veja também

ST_RasterToWorldCoordX, ST_RasterToWorldCoordY, ST_SetSkew

10.4.12 ST_RasterToWorldCoordX

ST_RasterToWorldCoordX — Retorna a coordenada geométrica X superior esquerda de um raster, coluna ou linha. A numeração das colunas e linhas começam no 1.

Synopsis

float8 **ST_RasterToWorldCoordX**(raster rast, integer xcolumn);
float8 **ST_RasterToWorldCoordX**(raster rast, integer xcolumn, integer yrow);

Descrição

Retorna a coordenada X superior esquerda de uma coluna raster em unidades geométricas do raster georreferenciado. A numeração da coluna e da linha começa no 1, mas se algum passar como zero, um número negativo ou um maior que a respectiva dimensão do raster, retornará as coordenas fora do raster para esquerda ou direita, assumindo que a skew e tamanhos do pixel são os mesmo que o raster selecionado.

Note
Para rasters sem desvio, fornecer a coluna X é suficiente. Para rasters com desvio, a coordenada georreferenciada é uma função da ST_ScaleX e ST_SkewX e linha e coluna. Um erro aparecerá se você der somente a coluna X para um raster desviado.

Alterações: 2.1.0 Em versões anteriores, era chamado de ST_Raster2WorldCoordX

Exemplos

```
-- non-skewed raster providing column is sufficient
SELECT rid, ST_RasterToWorldCoordX(rast,1) As x1coord,
    ST_RasterToWorldCoordX(rast,2) As x2coord,
    ST_ScaleX(rast) As pixelx
FROM dummy_rast;
```

```
 rid |  x1coord   |  x2coord  | pixelx
-----+------------+-----------+--------
   1 |        0.5 |       2.5 |      2
   2 | 3427927.75 | 3427927.8 |   0.05
```

```
-- for fun lets skew it
SELECT rid, ST_RasterToWorldCoordX(rast, 1, 1) As x1coord,
    ST_RasterToWorldCoordX(rast, 2, 3) As x2coord,
    ST_ScaleX(rast) As pixelx
FROM (SELECT rid, ST_SetSkew(rast, 100.5, 0) As rast FROM dummy_rast) As foo;

 rid |  x1coord   |  x2coord  | pixelx
-----+------------+-----------+--------
   1 |        0.5 |     203.5 |      2
   2 | 3427927.75 | 3428128.8 |   0.05
```

Veja também

ST_ScaleX, ST_RasterToWorldCoordY, ST_SetSkew, ST_SkewX

10.4.13 ST_RasterToWorldCoordY

ST_RasterToWorldCoordY — Retorna a coordenada geométrica Y superior esquerda de um raster, coluna e linha. A numeração das colunas e linhas começam no 1.

Synopsis

float8 **ST_RasterToWorldCoordY**(raster rast, integer yrow);
float8 **ST_RasterToWorldCoordY**(raster rast, integer xcolumn, integer yrow);

Descrição

Retorna a coordenada Y superior esquerda de uma coluna raster em unidades geométricas do raster georreferenciado. A numeração da coluna e da linha começa no 1, mas se algum passar como zero, um número negativo ou um maior que a respectiva dimensão do raster, retornará as coordenas fora do raster para esquerda ou direita, assumindo que o desvio e tamanhos do pixel são os mesmos que o raster selecionado.

Note
Para rasters sem desvio, fornecer a coluna Y é suficiente. Para rasters com desvio, a coordenada georreferenciada é uma função da ST_ScaleY e ST_SkewY e linha e coluna. Um erro aparecerá se você der somente a linha Y para um raster desviado.

Alterações: 2.1.0 Em versões anteriores, era chamado de ST_Raster2WorldCoordY

Exemplos

```
-- non-skewed raster providing row is sufficient
SELECT rid, ST_RasterToWorldCoordY(rast,1) As y1coord,
    ST_RasterToWorldCoordY(rast,3) As y2coord,
    ST_ScaleY(rast) As pixely
FROM dummy_rast;
```

```
 rid | y1coord |  y2coord  | pixely
-----+---------+-----------+--------
   1 |     0.5 |       6.5 |      3
   2 | 5793244 | 5793243.9 |  -0.05
```

```
-- for fun lets skew it
SELECT rid, ST_RasterToWorldCoordY(rast,1,1) As y1coord,
    ST_RasterToWorldCoordY(rast,2,3) As y2coord,
    ST_ScaleY(rast) As pixely
FROM (SELECT rid, ST_SetSkew(rast,0,100.5) As rast FROM dummy_rast) As foo;

 rid | y1coord |  y2coord  | pixely
-----+---------+-----------+--------
   1 |     0.5 |       107 |      3
   2 | 5793244 | 5793344.4 |  -0.05
```

Veja também

ST_ScaleY, ST_RasterToWorldCoordX, ST_SetSkew, ST_SkewY

10.4.14 ST_Rotation

ST_Rotation — Retorna a rotação do raster em radianos.

Synopsis

float8 **ST_Rotation**(raster rast);

Descrição

Retorna a rotação uniforme do raster em radianos. Se um raster n ão tiver uma rotação uniforme, NaN retorna. Recorra a World File para mais detalhes.

Exemplos

```
SELECT rid, ST_Rotation(ST_SetScale(ST_SetSkew(rast, sqrt(2)), sqrt(2))) as rot FROM  ↩
    dummy_rast;

 rid |        rot
-----+-------------------
   1 | 0.785398163397448
   2 | 0.785398163397448
```

Veja também

ST_SetRotation, ST_SetScale, ST_SetSkew

10.4.15 ST_SkewX

ST_SkewX — Retorna o desvio X georreferência (ou parâmetro e rotação).

Synopsis

float8 **ST_SkewX**(raster rast);

Descrição

Retorna o desvio X georreferência (ou parâmetro de rotação). Recorra a World File para mais detalhes.

Exemplos

```
SELECT rid, ST_SkewX(rast) As skewx, ST_SkewY(rast) As skewy,
    ST_GeoReference(rast) as georef
FROM dummy_rast;

 rid | skewx | skewy |       georef
-----+-------+-------+--------------------
   1 |     0 |     0 | 2.0000000000
                     : 0.0000000000
                     : 0.0000000000
                     : 3.0000000000
                     : 0.5000000000
                     : 0.5000000000
                     :
   2 |     0 |     0 | 0.0500000000
                     : 0.0000000000
                     : 0.0000000000
                     : -0.0500000000
                     : 3427927.7500000000
                     : 5793244.0000000000
```

Veja também

ST_GeoReference, ST_SkewY, ST_SetSkew

10.4.16 ST_SkewY

ST_SkewY — Retorna o desvio Y georreferência (ou parâmetro e rotação).

Synopsis

float8 **ST_SkewY**(raster rast);

Descrição

Retorna o desvio Y georreferência (ou parâmetro de rotação). Recorra a World File para mais detalhes.

Exemplos

```
SELECT rid, ST_SkewX(rast) As skewx, ST_SkewY(rast) As skewy,
    ST_GeoReference(rast) as georef
FROM dummy_rast;

 rid | skewx | skewy |       georef
```

```
-----+-------+-------+--------------------
   1 |     0 |     0 | 2.0000000000
                     : 0.0000000000
                     : 0.0000000000
                     : 3.0000000000
                     : 0.5000000000
                     : 0.5000000000
                     :
   2 |     0 |     0 | 0.0500000000
                     : 0.0000000000
                     : 0.0000000000
                     : -0.0500000000
                     : 3427927.7500000000
                     : 5793244.0000000000
```

Veja também

ST_GeoReference, ST_SkewX, ST_SetSkew

10.4.17 ST_SRID

ST_SRID — Retorna o identificador de referência espacial como definido na tabela spatial_ref_sys.

Synopsis

integer **ST_SRID**(raster rast);

Descrição

Retorna o identificador de referência espacial do objeto raster como definido na tabela spatial_ref_sys.

Note
A partir do PostGIS 2.0+ o srid de raster/geometria não georreferenciado é 0 em vez de -1.

Exemplos

```
SELECT ST_SRID(rast) As srid
FROM dummy_rast WHERE rid=1;

srid
----------------
0
```

Veja também

Section 4.5, ST_SRID

10.4.18 ST_Summary

ST_Summary — Retorna um texto resumo dos conteúdos do raster.

Synopsis

text **ST_Summary**(raster rast);

Descrição

Retorna um texto resumo dos conteúdos do raster.

Disponibilidade: 2.1.0

Exemplos

```
SELECT ST_Summary(
    ST_AddBand(
        ST_AddBand(
            ST_AddBand(
                ST_MakeEmptyRaster(10, 10, 0, 0, 1, -1, 0, 0, 0)
                , 1, '8BUI', 1, 0
            )
            , 2, '32BF', 0, -9999
        )
        , 3, '16BSI', 0, NULL
    )
);

                            st_summary
------------------------------------------------------------------
 Raster of 10x10 pixels has 3 bands and extent of BOX(0 -10,10 0)+
     band 1 of pixtype 8BUI is in-db with NODATA value of 0      +
     band 2 of pixtype 32BF is in-db with NODATA value of -9999  +
     band 3 of pixtype 16BSI is in-db with no NODATA value
(1 row)
```

Veja também

ST_MetaData, ST_BandMetaData, ST_Summary ST_Extent

10.4.19 ST_UpperLeftX

ST_UpperLeftX — Retorna a coordenada X superior esquerda na ref. espacial projetada.

Synopsis

float8 **ST_UpperLeftX**(raster rast);

Descrição

Retorna a coordenada X superior esquerda na ref. espacial projetada.

Exemplos

```
SELECt rid, ST_UpperLeftX(rast) As ulx
FROM dummy_rast;

 rid |    ulx
-----+------------
   1 |        0.5
   2 | 3427927.75
```

Veja também

ST_UpperLeftY, ST_GeoReference, Caixa3D

10.4.20 ST_UpperLeftY

ST_UpperLeftY — Retorna a coordenada Y superior esquerda na ref. espacial projetada.

Synopsis

float8 **ST_UpperLeftY**(raster rast);

Descrição

Retorna a coordenada Y superior esquerda na ref. espacial projetada.

Exemplos

```
SELECT rid, ST_UpperLeftY(rast) As uly
FROM dummy_rast;

 rid |   uly
-----+---------
   1 |     0.5
   2 | 5793244
```

Veja também

ST_UpperLeftX, ST_GeoReference, Caixa3D

10.4.21 ST_Width

ST_Width — Retorna a largura do raster em pixeis.

Synopsis

integer **ST_Width**(raster rast);

Descrição

Retorna a largura do raster em pixeis.

Exemplos

```
SELECT ST_Width(rast) As rastwidth
FROM dummy_rast WHERE rid=1;

rastwidth
----------------
10
```

Veja também

ST_Height

10.4.22 ST_WorldToRasterCoord

ST_WorldToRasterCoord — Retorna o canto superior esquerdo como coluna e linha dados os X e Y geométricos (longitude e latitude) ou um ponto expressado na coordenada do sistema de referência espacial do raster.

Synopsis

record **ST_WorldToRasterCoord**(raster rast, geometry pt);
record **ST_WorldToRasterCoord**(raster rast, double precision longitude, double precision latitude);

Descrição

Retorna o canto superior esquerdo como coluna e linha dados os X e Y geométricos (longitude e latitude) ou um ponto. Esta função funciona se o X e Y geométricos ou ponto estiver fora da extensão do raster ou não. O X e Y geométricos devem ser expressados na coordenada do sistema de referência espacial do raster.

Disponibilidade: 2.1.0

Exemplos

```
SELECT
    rid,
    (ST_WorldToRasterCoord(rast,3427927.8,20.5)).*,
    (ST_WorldToRasterCoord(rast,ST_GeomFromText('POINT(3427927.8 20.5)',ST_SRID(rast)))).*
FROM dummy_rast;

 rid | columnx |   rowy    | columnx |   rowy
-----+---------+-----------+---------+-----------
   1 | 1713964 |         7 | 1713964 |         7
   2 |       2 | 115864471 |       2 | 115864471
```

Veja também

ST_WorldToRasterCoordX, ST_WorldToRasterCoordY, ST_RasterToWorldCoordX, ST_RasterToWorldCoordY, ST_SRID

10.4.23 ST_WorldToRasterCoordX

ST_WorldToRasterCoordX — Retorna a coluna no raster do ponto (pt) ou uma coordenada X e Y (xw, yw) representada no sistema de referência espacial mundial de raster.

Synopsis

integer **ST_WorldToRasterCoordX**(raster rast, geometry pt);
integer **ST_WorldToRasterCoordX**(raster rast, double precision xw);
integer **ST_WorldToRasterCoordX**(raster rast, double precision xw, double precision yw);

Descrição

Retorna a coluna no raster do ponto (pt) ou uma coordenada X e Y (xw, yw). Um ponto, ou (ambas as coordenadas xw e yw são requeridas se um raster estiver desviado). Se um raster não estiver desviado, então xw é o suficiente. Coordenadas globais estão no sistema de coordenadas de referência espacial do raster.

Alterações: 2.1.0 Em versões anteriores, era chamado de ST_World2RasterCoordX

Exemplos

```
SELECT rid, ST_WorldToRasterCoordX(rast,3427927.8) As xcoord,
        ST_WorldToRasterCoordX(rast,3427927.8,20.5) As xcoord_xwyw,
        ST_WorldToRasterCoordX(rast,ST_GeomFromText('POINT(3427927.8 20.5)',ST_SRID(rast))) ↩
            As ptxcoord
FROM dummy_rast;

 rid | xcoord  |  xcoord_xwyw   | ptxcoord
-----+---------+---------+----------
   1 | 1713964 | 1713964 |  1713964
   2 |       1 |       1 |        1
```

Veja também

ST_RasterToWorldCoordX, ST_RasterToWorldCoordY, ST_SRID

10.4.24 ST_WorldToRasterCoordY

ST_WorldToRasterCoordY — Retorna a linha no raster do ponto (pt) ou uma coordenada X e Y (xw, yw) representada no sistema de referência espacial global de raster.

Synopsis

integer **ST_WorldToRasterCoordY**(raster rast, geometry pt);
integer **ST_WorldToRasterCoordY**(raster rast, double precision xw);
integer **ST_WorldToRasterCoordY**(raster rast, double precision xw, double precision yw);

Descrição

Retorna a linha no raster do ponto (pt) ou uma coordenada X e Y (xw, yw). Um ponto, ou (ambas as coordenadas xw e yw são requeridas se um raster estiver desviado). Se um raster não estiver desviado, então xw é o suficiente. Coordenadas globais estão no sistema de coordenadas de referência espacial do raster.

Alterações: 2.1.0 Em versões anteriores, era chamado de ST_World2RasterCoordY

Exemplos

```
SELECT rid, ST_WorldToRasterCoordY(rast,20.5) As ycoord,
        ST_WorldToRasterCoordY(rast,3427927.8,20.5) As ycoord_xwyw,
        ST_WorldToRasterCoordY(rast,ST_GeomFromText('POINT(3427927.8 20.5)',ST_SRID(rast))) ↩
            As ptycoord
FROM dummy_rast;

 rid |  ycoord   | ycoord_xwyw | ptycoord
-----+-----------+-------------+-----------
   1 |         7 |           7 |         7
   2 | 115864471 |   115864471 | 115864471
```

Veja também

ST_RasterToWorldCoordX, ST_RasterToWorldCoordY, ST_SRID

10.5 Assessores de banda raster

10.5.1 ST_BandMetaData

ST_BandMetaData — Retorna os metadados básicos para uma banda raster especificada. banda número 1 é assumida se nenhuma for especificada.

Synopsis

(1) record **ST_BandMetaData**(raster rast, integer band=1);
(2) record **ST_BandMetaData**(raster rast, integer[] band);

Descrição

Returns basic meta data about a raster band. Columns returned: pixeltype, nodatavalue, isoutdb, path, outdbbandnum, filesize, filetimestamp.

Note
Se o raster não contém nenhuma banda, então surge um erro.

Note
If band has no NODATA value, nodatavalue are NULL.

Note
If isoutdb is False, path, outdbbandnum, filesize and filetimestamp are NULL. If outdb access is disabled, filesize and filetimestamp will also be NULL.

Enhanced: 2.5.0 to include *outdbbandnum*, *filesize* and *filetimestamp* for outdb rasters.

Exemplos: Variante 1

```
SELECT
    rid,
    (foo.md).*
FROM (
    SELECT
        rid,
        ST_BandMetaData(rast, 1) AS md
    FROM dummy_rast
    WHERE rid=2
) As foo;

 rid | pixeltype | nodatavalue | isoutdb | path | outdbbandnum
-----+-----------+---- --------+---------+------+--------------
   2 | 8BUI      |           0 | f       |      |
```

Exemplos: Variant 2

```
WITH foo AS (
    SELECT
        ST_AddBand(NULL::raster, '/home/pele/devel/geo/postgis-git/raster/test/regress/ ↩
            loader/Projected.tif', NULL::int[]) AS rast
)
SELECT
    *
FROM ST_BandMetadata(
    (SELECT rast FROM foo),
    ARRAY[1,3,2]::int[]
);

 bandnum | pixeltype | nodatavalue | isoutdb |                                        path  ↩
                                          | outdbbandnum  | filesize | filetimestamp |
---------+-----------+-------------+---------+-----------------------------------------------------

       1 | 8BUI      |             | t       | /home/pele/devel/geo/postgis-git/raster/test ↩
           /regress/loader/Projected.tif      |            1  |    12345 |    1521807257 |
       3 | 8BUI      |             | t       | /home/pele/devel/geo/postgis-git/raster/test ↩
           /regress/loader/Projected.tif      |            3  |    12345 |    1521807257 |
       2 | 8BUI      |             | t       | /home/pele/devel/geo/postgis-git/raster/test ↩
           /regress/loader/Projected.tif      |            2  |    12345 |    1521807257 |
```

Veja também

ST_MetaData, ST_BandPixelType

10.5.2 ST_BandNoDataValue

ST_BandNoDataValue — Retorna o valor em uma dada banda que não representa nenhum valor. Se nenhuma banda número 1 for assumida.

Synopsis

double precision **ST_BandNoDataValue**(raster rast, integer bandnum=1);

Descrição

Retorna o valor que não representa nenhum dado para a banda

Exemplos

```
SELECT ST_BandNoDataValue(rast,1) As bnval1,
    ST_BandNoDataValue(rast,2) As bnval2, ST_BandNoDataValue(rast,3) As bnval3
FROM dummy_rast
WHERE rid = 2;

 bnval1 | bnval2 | bnval3
--------+--------+--------
      0 |      0 |      0
```

Veja também

ST_NumBands

10.5.3 ST_BandIsNoData

ST_BandIsNoData — Retorna verdadeiro se a banda estiver repleta somente de valores nodata.

Synopsis

boolean **ST_BandIsNoData**(raster rast, integer band, boolean forceChecking=true);
boolean **ST_BandIsNoData**(raster rast, boolean forceChecking=true);

Descrição

Retorna verdadeiro se a banda estiver repleta apenas de valores nodata. Banda 1 é assumida se não especificada. Se o último argumento for VERDADEIRO, a banda toda é verificada pixel por pixel. Caso contrário, a função simplesmente retorna o valor da bandeira isnodata para a banda. O valor padrão para este parâmetro é FALSO, se não especificado.

Disponibilidade: 2.0.0

Note
Se a bandeira for suja (ou seja, o resultado for diferente usando VERDADEIRO como último parâmetro e não usando ele) você deveria atualizar o raster para tornar esta bandeira verdadeira, usando a ST_SetBandIsNodata(), ou ST_SetBandNodataValue() com VERDADEIRO como último argumento. Veja ST_SetBandIsNoData.

Exemplos

```
-- Create dummy table with one raster column
create table dummy_rast (rid integer, rast raster);

-- Add raster with two bands, one pixel/band. In the first band, nodatavalue = pixel value
    = 3.
-- In the second band, nodatavalue = 13, pixel value = 4
insert into dummy_rast values(1,
(
'01' -- little endian (uint8 ndr)
```

```
||
'0000' -- version (uint16 0)
||
'0200' -- nBands (uint16 0)
||
'17263529ED684A3F' -- scaleX (float64 0.000805965234044584)
||
'F9253529ED684ABF' -- scaleY (float64 -0.00080596523404458)
||
'1C9F33CE69E352C0' -- ipX (float64 -75.5533328537098)
||
'718F0E9A27A44840' -- ipY (float64 49.2824585505576)
||
'ED50EB853EC32B3F' -- skewX (float64 0.000211812383858707)
||
'7550EB853EC32B3F' -- skewY (float64 0.000211812383858704)
||
'E6100000' -- SRID (int32 4326)
||
'0100' -- width (uint16 1)
||
'0100' -- height (uint16 1)
||
'6' -- hasnodatavalue and isnodata value set to true.
||
'2' -- first band type (4BUI)
||
'03' -- novalue==3
||
'03' -- pixel(0,0)==3 (same that nodata)
||
'0' -- hasnodatavalue set to false
||
'5' -- second band type (16BSI)
||
'0D00' -- novalue==13
||
'0400' -- pixel(0,0)==4
)::raster
);

select st_bandisnodata(rast, 1) from dummy_rast where rid = 1; -- Expected true
select st_bandisnodata(rast, 2) from dummy_rast where rid = 1; -- Expected false
```

Veja também

ST_BandNoDataValue, ST_NumBands, ST_SetBandNoDataValue, ST_SetBandIsNoData

10.5.4 ST_BandPath

ST_BandPath — Retorna o caminho do arquivo do sistema para uma banda armazenada em um sistema de arquivos. Se nenhum número de banda for especificado, usa-se 1.

Synopsis

text **ST_BandPath**(raster rast, integer bandnum=1);

Descrição

Retorna o caminho do arquivo do sistema para uma banda. Surge um erro se for chamado com uma banda no banco de dados.

Exemplos

Veja também

10.5.5 ST_BandFileSize

ST_BandFileSize — Returns the file size of a band stored in file system. If no bandnum specified, 1 is assumed.

Synopsis

bigint **ST_BandFileSize**(raster rast, integer bandnum=1);

Descrição

Returns the file size of a band stored in file system. Throws an error if called with an in db band, or if outdb access is not enabled.

This function is typically used in conjunction with ST_BandPath() and ST_BandFileTimestamp() so a client can determine if the filename of a outdb raster as seen by it is the same as the one seen by the server.

Availability: 2.5.0

Exemplos

```
SELECT ST_BandFileSize(rast,1) FROM dummy_rast WHERE rid = 1;

 st_bandfilesize
-----------------
          240574
```

10.5.6 ST_BandFileTimestamp

ST_BandFileTimestamp — Returns the file timestamp of a band stored in file system. If no bandnum specified, 1 is assumed.

Synopsis

bigint **ST_BandFileTimestamp**(raster rast, integer bandnum=1);

Descrição

Returns the file timestamp (number of seconds since Jan 1st 1970 00:00:00 UTC) of a band stored in file system. Throws an error if called with an in db band, or if outdb access is not enabled.

This function is typically used in conjunction with ST_BandPath() and ST_BandFileSize() so a client can determine if the filename of a outdb raster as seen by it is the same as the one seen by the server.

Availability: 2.5.0

Exemplos

```
SELECT ST_BandFileTimestamp(rast,1) FROM dummy_rast WHERE rid = 1;

 st_bandfiletimestamp
----------------------
           1521807257
```

10.5.7 ST_BandPixelType

ST_BandPixelType — Retorna o tipo pixel para uma dada banda. Se nenhum número de banda for especificado, usa-se 1.

Synopsis

text **ST_BandPixelType**(raster rast, integer bandnum=1);

Descrição

Returns name describing data type and size of values stored in each cell of given band.

Existem 11 tipos de pixel. Os tipos suportados são os seguintes:

- 1BB - 1-bit boolean
- 2BUI - 2-bit inteiro não assinado
- 4BUI - 4-bit inteiro não assinado
- 8BSI - 8-bit inteiro assinado
- 8BUI - 8-bit inteiro não assinado
- 16BSI - 16-bit inteiro assinado
- 16BUI - 16-bit inteiro não assinado
- 32BSI - 32-bit inteiro assinado
- 32BUI - 32-bit inteiro não assinado
- 32BF - 32-bit float
- 64BF - 64-bit float

Exemplos

```
SELECT ST_BandPixelType(rast,1) As btype1,
    ST_BandPixelType(rast,2) As btype2, ST_BandPixelType(rast,3) As btype3
FROM dummy_rast
WHERE rid = 2;

 btype1 | btype2 | btype3
--------+--------+--------
 8BUI   | 8BUI   | 8BUI
```

Veja também

ST_NumBands

10.5.8 ST_PixelOfValue

ST_PixelOfValue — Retorna o número de bandas no objeto raster.

Synopsis

integer **ST_MemSize**(raster rast);

Descrição

Retorna o número de bandas no objeto raster.

Exemplos

```
SELECT ST_MinPossibleValue('16BSI');

 st_minpossiblevalue
---------------------
              -32768


SELECT ST_MinPossibleValue('8BUI');

 st_minpossiblevalue
---------------------
                   0
```

Veja também

ST_BandPixelType

10.5.9 ST_HasNoBand

ST_HasNoBand — Retorna verdade se não existirem bandas com números dados. Se nenhum número de banda for especificado, então assume-se a banda 1.

Synopsis

boolean **ST_HasNoBand**(raster rast, integer bandnum=1);

Descrição

Retorna verdade se não existirem bandas com números dados. Se nenhum número de banda for especificado, então assume-se a banda 1.

Disponibilidade: 2.0.0

Exemplos

```
SELECT rid, ST_HasNoBand(rast) As hb1, ST_HasNoBand(rast,2) as hb2,
ST_HasNoBand(rast,4) as hb4, ST_NumBands(rast) As numbands
FROM dummy_rast;

rid | hb1 | hb2 | hb4 | numbands
-----+-----+-----+-----+----------
1 | t   | t   | t   |        0
2 | f   | f   | t   |        3
```

Veja também

ST_NumBands

10.6 Assessores e Setters de Pixel Raster

10.6.1 ST_PixelAsPolygon

ST_PixelAsPolygon — Retorna o polígono que limita o pixel para uma linha e coluna específicas.

Synopsis

geometry **ST_PixelAsPolygon**(raster rast, integer columnx, integer rowy);

Descrição

Retorna o polígono que limita o pixel para uma linha e coluna específicas.

Disponibilidade: 2.0.0

Exemplos

```
-- get raster pixel polygon
SELECT i,j, ST_AsText(ST_PixelAsPolygon(foo.rast, i,j)) As b1pgeom
FROM dummy_rast As foo
    CROSS JOIN generate_series(1,2) As i
    CROSS JOIN generate_series(1,1) As j
WHERE rid=2;

 i | j |                                                                                b1pgeom
---+---+-------------------------------------------------------------------------------------------------
 1 | 1 | POLYGON((3427927.75 5793244,3427927.8 5793244,3427927.8 5793243.95,...
 2 | 1 | POLYGON((3427927.8 5793244,3427927.85 5793244,3427927.85 5793243.95, ..
```

Veja também

ST_DumpAsPolygons, ST_PixelAsPolygons, ST_PixelAsPoint, ST_PixelAsPoints, ST_PixelAsCentroid, ST_PixelAsCentroids, ST_Intersection, ST_AsText

10.6.2 ST_PixelAsPolygons

ST_PixelAsPolygons — Retorna o polígono que limita cada pixel de uma banda raster ao longo do valor, as coordenadas raster X e Y de cada pixel.

Synopsis

setof record **ST_PixelAsPolygons**(raster rast, integer band=1, boolean exclude_nodata_value=TRUE);

Descrição

Retorna o polígono que limita cada pixel de uma banda raster ao longo do valor (precisão dobrada), as coordenadas (inteiras) raster X e Y de cada pixel.

Return record format: *geom* geometry, *val* double precision, *x* integer, *y* integers.

Note
When *exclude_nodata_value* = TRUE, only those pixels whose values are not NODATA are returned as points.

Note
ST_PixelAsPolygons retorna um polígono para cada pixel. Isto é diferente da ST_DumpAsPolygons, onde cada geometria representa um ou mais pixeis com o mesmo valor.

Disponibilidade: 2.0.0

Melhorias: 2.1.0 o argumento opcional exclude_nodata_value foi adicionado.

Alterações: 2.1.1 Mudança no comportamento do exclude_nodata_value.

Exemplos

```
-- get raster pixel polygon
SELECT (gv).x, (gv).y, (gv).val, ST_AsText((gv).geom) geom
FROM (SELECT ST_PixelAsPolygons(
                 ST_SetValue(ST_SetValue(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 0.001,  ↩
                     -0.001, 0.001, 0.001, 4269),
                                                 '8BUI'::text, 1, 0),
                                     2, 2, 10),
                             1, 1, NULL)
) gv
) foo;

 x | y | val |                geom
---+---+-----+-------------------------------------------------------------------------
 1 | 1 |     | POLYGON((0 0,0.001 0.001,0.002 0,0.001 -0.001,0 0))
 1 | 2 |   1 | POLYGON((0.001 -0.001,0.002 0,0.003 -0.001,0.002 -0.002,0.001 -0.001))
 2 | 1 |   1 | POLYGON((0.001 0.001,0.002 0.002,0.003 0.001,0.002 0,0.001 0.001))
 2 | 2 |  10 | POLYGON((0.002 0,0.003 0.001,0.004 0,0.003 -0.001,0.002 0))
```

Veja também

ST_DumpAsPolygons, ST_PixelAsPolygon, ST_PixelAsPoint, ST_PixelAsPoints, ST_PixelAsCentroid, ST_PixelAsCentroids, ST_AsText

10.6.3 ST_PixelAsPoint

ST_PixelAsPoint — Retorna um ponto geométrico do canto superior esquerdo do pixel.

Synopsis

geometry **ST_PixelAsPoint**(raster rast, integer columnx, integer rowy);

Descrição

Retorna um ponto geométrico do canto superior esquerdo do pixel.

Disponibilidade: 2.1.0

Exemplos

```
SELECT ST_AsText(ST_PixelAsPoint(rast, 1, 1)) FROM dummy_rast WHERE rid = 1;

   st_astext
----------------
 POINT(0.5 0.5)
```

Veja também

ST_DumpAsPolygons, ST_PixelAsPolygon, ST_PixelAsPolygons, ST_PixelAsPoints, ST_PixelAsCentroid, ST_PixelAsCentroids

10.6.4 ST_PixelAsPoints

ST_PixelAsPoints — Retorna um ponto geométrico para cada pixel de uma banda raster junto com o valor, as coordenas raster X e Y de cada pixel. As coordenadas do ponto são do ponto esquerdo superior do pixel.

Synopsis

setof record **ST_PixelAsPoints**(raster rast, integer band=1, boolean exclude_nodata_value=TRUE);

Descrição

Retorna um ponto geométrico para cada pixel de uma banda raster junto com o valor, as coordenas raster X e Y de cada pixel. As coordenadas do ponto são do ponto esquerdo superior do pixel.

Return record format: *geom* geometry, *val* double precision, *x* integer, *y* integers.

Note
When *exclude_nodata_value* = TRUE, only those pixels whose values are not NODATA are returned as points.

Disponibilidade: 2.1.0

Alterações: 2.1.1 Mudança no comportamento do exclude_nodata_value.

Exemplos

```
SELECT x, y, val, ST_AsText(geom) FROM (SELECT (ST_PixelAsPoints(rast, 1)).* FROM  ↩
    dummy_rast WHERE rid = 2) foo;

 x | y | val |          st_astext
---+---+-----+-----------------------------
 1 | 1 | 253 | POINT(3427927.75 5793244)
 2 | 1 | 254 | POINT(3427927.8 5793244)
 3 | 1 | 253 | POINT(3427927.85 5793244)
 4 | 1 | 254 | POINT(3427927.9 5793244)
 5 | 1 | 254 | POINT(3427927.95 5793244)
 1 | 2 | 253 | POINT(3427927.75 5793243.95)
 2 | 2 | 254 | POINT(3427927.8 5793243.95)
 3 | 2 | 254 | POINT(3427927.85 5793243.95)
 4 | 2 | 253 | POINT(3427927.9 5793243.95)
 5 | 2 | 249 | POINT(3427927.95 5793243.95)
 1 | 3 | 250 | POINT(3427927.75 5793243.9)
 2 | 3 | 254 | POINT(3427927.8 5793243.9)
 3 | 3 | 254 | POINT(3427927.85 5793243.9)
 4 | 3 | 252 | POINT(3427927.9 5793243.9)
 5 | 3 | 249 | POINT(3427927.95 5793243.9)
 1 | 4 | 251 | POINT(3427927.75 5793243.85)
 2 | 4 | 253 | POINT(3427927.8 5793243.85)
 3 | 4 | 254 | POINT(3427927.85 5793243.85)
 4 | 4 | 254 | POINT(3427927.9 5793243.85)
 5 | 4 | 253 | POINT(3427927.95 5793243.85)
 1 | 5 | 252 | POINT(3427927.75 5793243.8)
 2 | 5 | 250 | POINT(3427927.8 5793243.8)
 3 | 5 | 254 | POINT(3427927.85 5793243.8)
 4 | 5 | 254 | POINT(3427927.9 5793243.8)
 5 | 5 | 254 | POINT(3427927.95 5793243.8)
```

Veja também

ST_DumpAsPolygons, ST_PixelAsPolygon, ST_PixelAsPolygons, ST_PixelAsPoint, ST_PixelAsCentroid, ST_PixelAsCentroids

10.6.5 ST_PixelAsCentroid

ST_PixelAsCentroid — Retorna o centroide (ponto) da área representada por um pixel.

Synopsis

geometry **ST_PixelAsCentroid**(raster rast, integer x, integer y);

Descrição

Retorna o centroide (ponto) da área representada por um pixel.

Melhorias: 2.1.0 Reescrito em C

Disponibilidade: 2.1.0

Exemplos

```
SELECT ST_AsText(ST_PixelAsCentroid(rast, 1, 1)) FROM dummy_rast WHERE rid = 1;

  st_astext
--------------
 POINT(1.5 2)
```

Veja também

ST_DumpAsPolygons, ST_PixelAsPolygon, ST_PixelAsPolygons, ST_PixelAsPoint, ST_PixelAsPoints, ST_PixelAsCentroids

10.6.6 ST_PixelAsCentroids

ST_PixelAsCentroids — Retorna o centroide (ponto geométrico) para cada pixel de uma banda raster junto com o valor, as coordenas raster X e Y de cada pixel. O ponto é o centroide da área representada por um pixel.

Synopsis

setof record **ST_PixelAsCentroids**(raster rast, integer band=1, boolean exclude_nodata_value=TRUE);

Descrição

Retorna o centroide (ponto geométrico) para cada pixel de uma banda raster junto com o valor, as coordenas raster X e Y de cada pixel. O ponto é o centroide da área representada por um pixel.

Return record format: *geom* geometry, *val* double precision, *x* integer, *y* integers.

Note

When *exclude_nodata_value* = TRUE, only those pixels whose values are not NODATA are returned as points.

Melhorias: 2.1.0 Reescrito em C

Alterações: 2.1.1 Mudança no comportamento do exclude_nodata_value.

Disponibilidade: 2.1.0

Exemplos

```
--LATERAL syntax requires PostgreSQL 9.3+
SELECT x, y, val, ST_AsText(geom)
    FROM (SELECT dp.* FROM dummy_rast, LATERAL ST_PixelAsCentroids(rast, 1) AS dp WHERE rid ↩
        = 2) foo;
 x | y | val |           st_astext
---+---+-----+-------------------------------
 1 | 1 | 253 | POINT(3427927.775 5793243.975)
 2 | 1 | 254 | POINT(3427927.825 5793243.975)
 3 | 1 | 253 | POINT(3427927.875 5793243.975)
 4 | 1 | 254 | POINT(3427927.925 5793243.975)
 5 | 1 | 254 | POINT(3427927.975 5793243.975)
 1 | 2 | 253 | POINT(3427927.775 5793243.925)
 2 | 2 | 254 | POINT(3427927.825 5793243.925)
 3 | 2 | 254 | POINT(3427927.875 5793243.925)
```

```
4 | 2 | 253 | POINT(3427927.925 5793243.925)
5 | 2 | 249 | POINT(3427927.975 5793243.925)
1 | 3 | 250 | POINT(3427927.775 5793243.875)
2 | 3 | 254 | POINT(3427927.825 5793243.875)
3 | 3 | 254 | POINT(3427927.875 5793243.875)
4 | 3 | 252 | POINT(3427927.925 5793243.875)
5 | 3 | 249 | POINT(3427927.975 5793243.875)
1 | 4 | 251 | POINT(3427927.775 5793243.825)
2 | 4 | 253 | POINT(3427927.825 5793243.825)
3 | 4 | 254 | POINT(3427927.875 5793243.825)
4 | 4 | 254 | POINT(3427927.925 5793243.825)
5 | 4 | 253 | POINT(3427927.975 5793243.825)
1 | 5 | 252 | POINT(3427927.775 5793243.775)
2 | 5 | 250 | POINT(3427927.825 5793243.775)
3 | 5 | 254 | POINT(3427927.875 5793243.775)
4 | 5 | 254 | POINT(3427927.925 5793243.775)
5 | 5 | 254 | POINT(3427927.975 5793243.775)
```

Veja também

ST_DumpAsPolygons, ST_PixelAsPolygon, ST_PixelAsPolygons, ST_PixelAsPoint, ST_PixelAsPoints, ST_PixelAsCentroid

10.6.7 ST_Value

ST_Value — Retorna o valor da banda dada com a colunax, linhay pixel ou em um ponto específico. Os números de banda começam em 1 e assumem-se 1 se não especificados. Se `exclude_nodata_value` for falso, então todos os pixeis, inclusive os `nodata`, são considerados para intersectar e retornar valor. Se `exclude_nodata_value` não passar então lê dos metadados do raster.

Synopsis

double precision **ST_Value**(raster rast, geometry pt, boolean exclude_nodata_value=true);
double precision **ST_Value**(raster rast, integer band, geometry pt, boolean exclude_nodata_value=true, text resample='nearest');
double precision **ST_Value**(raster rast, integer x, integer y, boolean exclude_nodata_value=true);
double precision **ST_Value**(raster rast, integer band, integer x, integer y, boolean exclude_nodata_value=true);

Descrição

Retorna o raster modificado resultante do valor de uma banda em uma dada colunax, linhay pixel ou os pixeis que intersectam uma geometria específica. Os números de banda começam no 1 e são assumidos como 1 se não estiverem especificados.

Se `exclude_nodata_value` for verdade, contará apenas pixeis com valor diferente do valor `nodata` do raster. `exclude_nodata` é falso para contar todos os pixeis.

The allowed values of the `resample` parameter are "nearest" which performs the default nearest-neighbor resampling, and "bilinear" which performs a bilinear interpolation to estimate the value between pixel centers.

Melhorias: 2.1.0 o argumento opcional exclude_nodata_value foi adicionado.

Melhorias: 2.0.0 o argumento opcional exclude_nodata_value foi adicionado.

Exemplos

```
-- get raster values at particular postgis geometry points
-- the srid of your geometry should be same as for your raster
SELECT rid, ST_Value(rast, foo.pt_geom) As b1pval, ST_Value(rast, 2, foo.pt_geom) As b2pval
FROM dummy_rast CROSS JOIN (SELECT ST_SetSRID(ST_Point(3427927.77, 5793243.76), 0) As ↩
    pt_geom) As foo
WHERE rid=2;

 rid | b1pval | b2pval
-----+--------+--------
   2 |    252 |     79


-- general fictitious example using a real table
SELECT rid, ST_Value(rast, 3, sometable.geom) As b3pval
FROM sometable
WHERE ST_Intersects(rast,sometable.geom);
```

```
SELECT rid, ST_Value(rast, 1, 1, 1) As b1pval,
    ST_Value(rast, 2, 1, 1) As b2pval, ST_Value(rast, 3, 1, 1) As b3pval
FROM dummy_rast
WHERE rid=2;

 rid | b1pval | b2pval | b3pval
-----+--------+--------+--------
   2 |    253 |     78 |     70
```

```
--- Get all values in bands 1,2,3 of each pixel --
SELECT x, y, ST_Value(rast, 1, x, y) As b1val,
    ST_Value(rast, 2, x, y) As b2val, ST_Value(rast, 3, x, y) As b3val
FROM dummy_rast CROSS JOIN
generate_series(1, 1000) As x CROSS JOIN generate_series(1, 1000) As y
WHERE rid =  2 AND x <= ST_Width(rast) AND y <= ST_Height(rast);

 x | y | b1val | b2val | b3val
---+---+-------+-------+-------
 1 | 1 |   253 |    78 |    70
 1 | 2 |   253 |    96 |    80
 1 | 3 |   250 |    99 |    90
 1 | 4 |   251 |    89 |    77
 1 | 5 |   252 |    79 |    62
 2 | 1 |   254 |    98 |    86
 2 | 2 |   254 |   118 |   108
 :
 :
```

```
--- Get all values in bands 1,2,3 of each pixel same as above but returning the upper left ↩
    point point of each pixel --
SELECT ST_AsText(ST_SetSRID(
    ST_Point(ST_UpperLeftX(rast) + ST_ScaleX(rast)*x,
        ST_UpperLeftY(rast) + ST_ScaleY(rast)*y),
        ST_SRID(rast))) As uplpt
    , ST_Value(rast, 1, x, y) As b1val,
    ST_Value(rast, 2, x, y) As b2val, ST_Value(rast, 3, x, y) As b3val
FROM dummy_rast CROSS JOIN
generate_series(1,1000) As x CROSS JOIN generate_series(1,1000) As y
WHERE rid =  2 AND x <= ST_Width(rast) AND y <= ST_Height(rast);

            uplpt            | b1val | b2val | b3val
-----------------------------+-------+-------+-------
 POINT(3427929.25 5793245.5) |   253 |    78 |    70
```

```
 POINT(3427929.25 5793247)   |   253 |    96 |    80
 POINT(3427929.25 5793248.5) |   250 |    99 |    90
:
```

```
--- Get a polygon formed by union of all pixels
    that fall in a particular value range and intersect particular polygon --
SELECT ST_AsText(ST_Union(pixpolyg)) As shadow
FROM (SELECT ST_Translate(ST_MakeEnvelope(
        ST_UpperLeftX(rast), ST_UpperLeftY(rast),
            ST_UpperLeftX(rast) + ST_ScaleX(rast),
            ST_UpperLeftY(rast) + ST_ScaleY(rast), 0
            ), ST_ScaleX(rast)*x, ST_ScaleY(rast)*y
        ) As pixpolyg, ST_Value(rast, 2, x, y) As b2val
    FROM dummy_rast CROSS JOIN
generate_series(1,1000) As x CROSS JOIN generate_series(1,1000) As y
WHERE rid =  2
    AND x <= ST_Width(rast) AND y <= ST_Height(rast)) As foo
WHERE
    ST_Intersects(
        pixpolyg,
        ST_GeomFromText('POLYGON((3427928 5793244,3427927.75 5793243.75,3427928  ↩
            5793243.75,3427928 5793244))',0)
        ) AND b2val != 254;


        shadow
--------------------------------------------------------------------------------
 MULTIPOLYGON(((3427928 5793243.9,3427928 5793243.85,3427927.95 5793243.85,3427927.95  ↩
     5793243.9,
 3427927.95 5793243.95,3427928 5793243.95,3427928.05 5793243.95,3427928.05  ↩
     5793243.9,3427928 5793243.9)),((3427927.95 5793243.9,3427927.95 579324
3.85,3427927.9 5793243.85,3427927.85 5793243.85,3427927.85 5793243.9,3427927.9  ↩
    5793243.9,3427927.9 5793243.95,
3427927.95 5793243.95,3427927.95 5793243.9)),((3427927.85 5793243.75,3427927.85  ↩
    5793243.7,3427927.8 5793243.7,3427927.8 5793243.75
,3427927.8 5793243.8,3427927.8 5793243.85,3427927.85 5793243.85,3427927.85  ↩
    5793243.8,3427927.85 5793243.75)),
((3427928.05 5793243.75,3427928.05 5793243.7,3427928 5793243.7,3427927.95  ↩
    5793243.7,3427927.95 5793243.75,3427927.95 5793243.8,3427
927.95 5793243.85,3427928 5793243.85,3427928 5793243.8,3427928.05 5793243.8,
3427928.05 5793243.75)),((3427927.95 5793243.75,3427927.95 5793243.7,3427927.9  ↩
    5793243.7,3427927.85 5793243.7,
3427927.85 5793243.75,3427927.85 5793243.8,3427927.85 5793243.85,3427927.9 5793243.85,
3427927.95 5793243.85,3427927.95 5793243.8,3427927.95 5793243.75)))
```

```
--- Checking all the pixels of a large raster tile can take a long time.
--- You can dramatically improve speed at some lose of precision by orders of magnitude
--  by sampling pixels using the step optional parameter of generate_series.
--  This next example does the same as previous but by checking 1 for every 4 (2x2) pixels  ↩
    and putting in the last checked
--  putting in the checked pixel as the value for subsequent 4

SELECT ST_AsText(ST_Union(pixpolyg)) As shadow
FROM (SELECT ST_Translate(ST_MakeEnvelope(
        ST_UpperLeftX(rast), ST_UpperLeftY(rast),
            ST_UpperLeftX(rast) + ST_ScaleX(rast)*2,
            ST_UpperLeftY(rast) + ST_ScaleY(rast)*2, 0
            ), ST_ScaleX(rast)*x, ST_ScaleY(rast)*y
        ) As pixpolyg, ST_Value(rast, 2, x, y) As b2val
    FROM dummy_rast CROSS JOIN
generate_series(1,1000,2) As x CROSS JOIN generate_series(1,1000,2) As y
```

```
WHERE rid =  2
    AND x <= ST_Width(rast)  AND y <= ST_Height(rast)  ) As foo
WHERE
    ST_Intersects(
        pixpolyg,
        ST_GeomFromText('POLYGON((3427928 5793244,3427927.75 5793243.75,3427928  ↩
            5793243.75,3427928 5793244))',0)
        ) AND b2val != 254;

        shadow
-------------------------------------------------------------------------------
 MULTIPOLYGON(((3427927.9 5793243.85,3427927.8 5793243.85,3427927.8 5793243.95,
 3427927.9 5793243.95,3427928 5793243.95,3427928.1 5793243.95,3427928.1 5793243.85,3427928  ↩
    5793243.85,3427927.9 5793243.85)),
 ((3427927.9 5793243.65,3427927.8 5793243.65,3427927.8 5793243.75,3427927.8  ↩
    5793243.85,3427927.9 5793243.85,
 3427928 5793243.85,3427928 5793243.75,3427928.1 5793243.75,3427928.1 5793243.65,3427928  ↩
    5793243.65,3427927.9 5793243.65)))
```

Veja também

ST_SetValue, ST_DumpAsPolygons, ST_NumBands, ST_PixelAsPolygon, ST_ScaleX, ST_ScaleY, ST_UpperLeftX, ST_UpperLeftY, ST_SRID, ST_AsText, ST_Point, ST_MakeEnvelope, ST_Intersects, ST_Intersection

10.6.8 ST_NearestValue

ST_NearestValue — Retorna o valor não-NODATA mais próximo de um dado pixel de banda especificado por uma colunax e linhay ou um ponto geométrico expressado no mesmo sistema de coordenada referência do raster.

Synopsis

double precision **ST_NearestValue**(raster rast, integer bandnum, geometry pt, boolean exclude_nodata_value=true);
double precision **ST_NearestValue**(raster rast, geometry pt, boolean exclude_nodata_value=true);
double precision **ST_NearestValue**(raster rast, integer bandnum, integer columnx, integer rowy, boolean exclude_nodata_value=true);
double precision **ST_NearestValue**(raster rast, integer columnx, integer rowy, boolean exclude_nodata_value=true);

Descrição

Returns the nearest non-NODATA value of a given band in a given columnx, rowy pixel or at a specific geometric point. If the columnx, rowy pixel or the pixel at the specified geometric point is NODATA, the function will find the nearest pixel to the columnx, rowy pixel or geometric point whose value is not NODATA.

O número de banda começa no 1 e `bandnum` é assumido a ser 1 se não estiver especificado. Se `exclude_nodata_value` for falso, então todos os pixeis inclusive os pixeis `nodata` são considerados para intersectar e retornar valor. Se `exclude_nodata_valu` não passar então lê dos metadados do raster.

Disponibilidade: 2.1.0

Note
ST_NearestValue é uma substituição drop-in para ST_Value.

Exemplos

```
-- pixel 2x2 has value
SELECT
    ST_Value(rast, 2, 2) AS value,
    ST_NearestValue(rast, 2, 2) AS nearestvalue
FROM (
    SELECT
        ST_SetValue(
            ST_SetValue(
                ST_SetValue(
                    ST_SetValue(
                        ST_SetValue(
                            ST_AddBand(
                                ST_MakeEmptyRaster(5, 5, -2, 2, 1, -1, 0, 0, 0),
                                '8BUI'::text, 1, 0
                            ),
                            1, 1, 0.
                        ),
                        2, 3, 0.
                    ),
                    3, 5, 0.
                ),
                4, 2, 0.
            ),
            5, 4, 0.
        ) AS rast
) AS foo

 value | nearestvalue
-------+--------------
     1 |            1
```

```
-- pixel 2x3 is NODATA
SELECT
    ST_Value(rast, 2, 3) AS value,
    ST_NearestValue(rast, 2, 3) AS nearestvalue
FROM (
    SELECT
        ST_SetValue(
            ST_SetValue(
                ST_SetValue(
                    ST_SetValue(
                        ST_SetValue(
                            ST_AddBand(
                                ST_MakeEmptyRaster(5, 5, -2, 2, 1, -1, 0, 0, 0),
                                '8BUI'::text, 1, 0
                            ),
                            1, 1, 0.
                        ),
                        2, 3, 0.
                    ),
                    3, 5, 0.
                ),
                4, 2, 0.
            ),
            5, 4, 0.
        ) AS rast
) AS foo

 value | nearestvalue
```

```
-------+--------------
       |            1
```

Veja também

ST_Neighborhood, ST_Value

10.6.9 ST_SetSkew

ST_SetSkew — Returns a geometry with the same X/Y coordinates as the input geometry, and values from the raster copied into the Z dimension using the requested resample algorithm.

Synopsis

bytea **ST_AsGDALRaster**(raster rast, text format, text[] options=NULL, integer srid=sameassource);

Descrição

Returns a geometry with the same X/Y coordinates as the input geometry, and values from the raster copied into the Z dimensions using the requested resample algorithm.

The `resample` parameter can be set to "nearest" to copy the values from the cell each vertex falls within, or "bilinear" to use bilinear interpolation to calculate a value that takes neighboring cells into account also.

Disponibilidade: 2.2.0

Exemplos

```
--
-- 2x2 test raster with values
--
-- 10 50
-- 40 20
--
WITH test_raster AS (
SELECT
ST_SetValues(
  ST_AddBand(
    ST_MakeEmptyRaster(width =
> 2, height =
> 2,
      upperleftx =
> 0, upperlefty =
> 2,
      scalex =
> 1.0, scaley =
> -1.0,
      skewx =
> 0, skewy =
> 0, srid =
> 4326),
    index =
> 1, pixeltype =
> '16BSI',
    initialvalue =
> 0,
```

```
    nodataval =
> -999),
  1,1,1,
  newvalueset =
>ARRAY[ARRAY[10.0::float8, 50.0::float8], ARRAY[40.0::float8, 20.0::float8]]) AS rast
)
SELECT
ST_AsText(
  ST_SetZ(
    rast,
    band =
> 1,
    geom =
> 'SRID=4326;LINESTRING(1.0 1.9, 1.0 0.2)'::geometry,
    resample =
> 'bilinear'
))
FROM test_raster

            st_astext
-----------------------------------
 LINESTRING Z (1 1.9 38,1 0.2 27)
```

Veja também

ST_Value, ST_SetSRID

10.6.10 ST_SetSkew

ST_SetSkew — Returns a geometry with the same X/Y coordinates as the input geometry, and values from the raster copied into the M dimension using the requested resample algorithm.

Synopsis

bytea **ST_AsGDALRaster**(raster rast, text format, text[] options=NULL, integer srid=sameassource);

Descrição

Returns a geometry with the same X/Y coordinates as the input geometry, and values from the raster copied into the M dimensions using the requested resample algorithm.

The `resample` parameter can be set to "nearest" to copy the values from the cell each vertex falls within, or "bilinear" to use bilinear interpolation to calculate a value that takes neighboring cells into account also.

Disponibilidade: 2.2.0

Exemplos

```
--
-- 2x2 test raster with values
--
-- 10 50
-- 40 20
--
WITH test_raster AS (
SELECT
```

```
ST_SetValues(
  ST_AddBand(
    ST_MakeEmptyRaster(width =
> 2, height =
> 2,
      upperleftx =
> 0, upperlefty =
> 2,
      scalex =
> 1.0, scaley =
> -1.0,
      skewx =
> 0, skewy =
> 0, srid =
> 4326),
    index =
> 1, pixeltype =
> '16BSI',
    initialvalue =
> 0,
    nodataval =
> -999),
  1,1,1,
  newvalueset =
>ARRAY[ARRAY[10.0::float8, 50.0::float8], ARRAY[40.0::float8, 20.0::float8]]) AS rast
)
SELECT
ST_AsText(
  ST_SetM(
    rast,
    band =
> 1,
    geom =
> 'SRID=4326;LINESTRING(1.0 1.9, 1.0 0.2)'::geometry,
    resample =
> 'bilinear'
))
FROM test_raster

            st_astext
---------------------------------
 LINESTRING M (1 1.9 38,1 0.2 27)
```

Veja também

ST_Value, ST_SetSRID

10.6.11 ST_Neighborhood

ST_Neighborhood — Retorna um arranjo de precisão 2-D dobrada dos valores não-NODATA em torno da banda de pixel especificada ou por uma colunaX e linhaY ou um ponto geométrico expressado no mesmo sistema de coordenada de referência especial como o raster.

Synopsis

double precision[][] **ST_Neighborhood**(raster rast, integer bandnum, integer columnX, integer rowY, integer distanceX, integer distanceY, boolean exclude_nodata_value=true);

double precision[][] **ST_Neighborhood**(raster rast, integer columnX, integer rowY, integer distanceX, integer distanceY, boolean exclude_nodata_value=true);
double precision[][] **ST_Neighborhood**(raster rast, integer bandnum, geometry pt, integer distanceX, integer distanceY, boolean exclude_nodata_value=true);
double precision[][] **ST_Neighborhood**(raster rast, geometry pt, integer distanceX, integer distanceY, boolean exclude_nodata_value=tru

Descrição

Retorna um arranjo de precisão 2-D dobrada dos valores não-`NODATA` em torno da banda de pixel especificada ou por uma colunaX e linhaY ou um ponto geométrico expressado no mesmo sistema de coordenada de referência especial como o raster. Os parâmetros `distanceX` e `distanceY` definem o número de pixeis em torno do pixel especificado nos eixos X e Y, ex.: Quero todos os valores dentro de uma distância de 3 pixeis no eixo X e 2 pixeis de distância no eixo Y ao redor do meu pixel de interesse. O valor central do arranjo 2-D será o valor no pixel especificado pela colunaX e linhaY ou ponto geométrico.

O número de banda começa no 1 e `bandnum` é assumido a ser 1 se não estiver especificado. Se `exclude_nodata_value` for falso, então todos os pixeis inclusive os pixeis `nodata` são considerados para intersectar e retornar valor. Se `exclude_nodata_valu` não passar então lê dos metadados do raster.

Note
O número de elementos ao longo de cada eixo do arranjo que está retornando 2-D é 2 * (`distanceX`|`distanceY`) + 1. Então, para uma `distanceX` e `distanceY` de 1, o arranjo que retorna será 3x3.

Note
A saída do arranjo 2-D pode ser passado para qualquer um dos processos raster de funções bultiin, ex.: ST_Min4ma, ST_Sum4ma, ST_Mean4ma.

Disponibilidade: 2.1.0

Exemplos

```
-- pixel 2x2 has value
SELECT
    ST_Neighborhood(rast, 2, 2, 1, 1)
FROM (
    SELECT
        ST_SetValues(
            ST_AddBand(
                ST_MakeEmptyRaster(5, 5, -2, 2, 1, -1, 0, 0, 0),
                '8BUI'::text, 1, 0
            ),
            1, 1, 1, ARRAY[
                [0, 1, 1, 1, 1],
                [1, 1, 1, 0, 1],
                [1, 0, 1, 1, 1],
                [1, 1, 1, 1, 0],
                [1, 1, 0, 1, 1]
            ]::double precision[],
            1
        ) AS rast
) AS foo

         st_neighborhood
--------------------------------
{{NULL,1,1},{1,1,1},{1,NULL,1}}
```

```
-- pixel 2x3 is NODATA
SELECT
    ST_Neighborhood(rast, 2, 3, 1, 1)
FROM (
    SELECT
        ST_SetValues(
            ST_AddBand(
                ST_MakeEmptyRaster(5, 5, -2, 2, 1, -1, 0, 0, 0),
                '8BUI'::text, 1, 0
            ),
            1, 1, 1, ARRAY[
                [0, 1, 1, 1, 1],
                [1, 1, 1, 0, 1],
                [1, 0, 1, 1, 1],
                [1, 1, 1, 1, 0],
                [1, 1, 0, 1, 1]
            ]::double precision[],
            1
        ) AS rast
) AS foo

       st_neighborhood
------------------------------
 {{1,1,1},{1,NULL,1},{1,1,1}}
```

```
-- pixel 3x3 has value
-- exclude_nodata_value = FALSE
SELECT
    ST_Neighborhood(rast, 3, 3, 1, 1, false)
FROM ST_SetValues(
            ST_AddBand(
                ST_MakeEmptyRaster(5, 5, -2, 2, 1, -1, 0, 0, 0),
                '8BUI'::text, 1, 0
            ),
            1, 1, 1, ARRAY[
                [0, 1, 1, 1, 1],
                [1, 1, 1, 0, 1],
                [1, 0, 1, 1, 1],
                [1, 1, 1, 1, 0],
                [1, 1, 0, 1, 1]
            ]::double precision[],
            1
        ) AS rast

      st_neighborhood
---------------------------
{{1,1,0},{0,1,1},{1,1,1}}
```

Veja também

ST_NearestValue, ST_Min4ma, ST_Max4ma, ST_Sum4ma, ST_Mean4ma, ST_Range4ma, ST_Distinct4ma, ST_StdDev4ma

10.6.12 ST_SetValue

ST_SetValue — Retorna o raster modificado resultante do valor de uma banda em uma dada colunax, linhay pixel ou os pixeis que intersectam uma geometria específica. Os números de banda começam no 1 e são assumidos como 1 se não estiverem especificados.

Synopsis

raster **ST_SetValue**(raster rast, integer bandnum, geometry geom, double precision newvalue);
raster **ST_SetValue**(raster rast, geometry geom, double precision newvalue);
raster **ST_SetValue**(raster rast, integer bandnum, integer columnx, integer rowy, double precision newvalue);
raster **ST_SetValue**(raster rast, integer columnx, integer rowy, double precision newvalue);

Descrição

Returns modified raster resulting from setting the specified pixels' values to new value for the designated band given the raster's row and column or a geometry. If no band is specified, then band 1 is assumed.

Melhorias: 2.1.0 Variante geométrica ST_SetValue() agora suporta qualquer tipo de geometria, não apenas ponto. A variante geométrica é um envoltório em torno da variante geomval[] da ST_SetValues()

Exemplos

```
-- Geometry example
SELECT (foo.geomval).val, ST_AsText(ST_Union((foo.geomval).geom))
FROM (SELECT ST_DumpAsPolygons(
        ST_SetValue(rast,1,
                ST_Point(3427927.75, 5793243.95),
                50)
            ) As geomval
FROM dummy_rast
where rid = 2) As foo
WHERE (foo.geomval).val < 250
GROUP BY (foo.geomval).val;

 val |                                                                st_astext
-----+---------------------------------------------------------------------------
  50 | POLYGON((3427927.75 5793244,3427927.75 5793243.95,3427927.8 579324 ...
 249 | POLYGON((3427927.95 5793243.95,3427927.95 5793243.85,3427928 57932 ...
```

```
-- Store the changed raster --
    UPDATE dummy_rast SET rast = ST_SetValue(rast,1, ST_Point(3427927.75, 5793243.95),100)
        WHERE rid = 2   ;
```

Veja também

ST_Value, ST_DumpAsPolygons

10.6.13 ST_SetValues

ST_SetValues — Retorna o raster modificado resultante dos valores de uma dada banda.

Synopsis

raster **ST_SetValues**(raster rast, integer nband, integer columnx, integer rowy, double precision[][] newvalueset, boolean[][] noset=NULL, boolean keepnodata=FALSE);
raster **ST_SetValues**(raster rast, integer nband, integer columnx, integer rowy, double precision[][] newvalueset, double precision nosetvalue, boolean keepnodata=FALSE);

raster **ST_SetValues**(raster rast, integer nband, integer columnx, integer rowy, integer width, integer height, double precision newvalue, boolean keepnodata=FALSE);
raster **ST_SetValues**(raster rast, integer columnx, integer rowy, integer width, integer height, double precision newvalue, boolean keepnodata=FALSE);
raster **ST_SetValues**(raster rast, integer nband, geomval[] geomvalset, boolean keepnodata=FALSE);

Descrição

Retorna o raster modificado resultante dos valores especificados do pixel para novo valor(es) para a banda designada.

Se `keepnodata` for VERDADE, aqueles pixeis cujos valores são NODATA não terão o valor correspondente em `newvalueset`.

Para Variante 1, os pixeis específicos são determinados pela `columnx`, `rowy` coordenadas pixel e as dimensões do arranjo `newvalueset`. `noset` pode ser usado para prevenir pixeis com valores presentes no `newvalueset` de serem estabelecidos (PostgreSQL não permitindo arranjos ragged/jagged). Veja o exemplo de Variante 1.

Variante 2 é como a Variante 1, mas com uma precisão dupla simples `nosetvalue` em vez de um arranjo booleano `noset`. Elementos no `newvalueset` com o valor `nosetvalue` são pulados. Veja o exemplo da Variante 2.

Para Variante 3, os pixeis a serem estabelecidos são determinados pelas `columnx`, `rowy` coordenadas pixel, `width` e `height`. Veja o exemplo da Variante 3.

A Variante 4 é a mesma que a Variante 3, com a exceção de que ela assume que a primeira banda do pixel de `rast` será estabelecida.

Para a Variante 5, um arranjo de geomval é usado para determinar os pixeis específicos. Se todas as geometrias no arranjo forem do tipo PONTO ou MULTIPONTO, a função usa um atalho onde a longitude e latitude de cada ponto é usada para pôr um pixel diretamente. Caso contrário, as geometrias são convertidas para rasters e então iteradas através de um passo. Veja o exemplo de Variante 5.

Disponibilidade: 2.1.0

Exemplos: Variante 1

```
/*
The ST_SetValues() does the following...

+ - + - + - +          + - + - + - +
| 1 | 1 | 1 |          | 1 | 1 | 1 |
+ - + - + - +          + - + - + - +
| 1 | 1 | 1 |    =
>    | 1 | 9 | 9 |
+ - + - + - +          + - + - + - +
| 1 | 1 | 1 |          | 1 | 9 | 9 |
+ - + - + - +          + - + - + - +
*/
SELECT
    (poly).x,
    (poly).y,
    (poly).val
FROM (
SELECT
    ST_PixelAsPolygons(
        ST_SetValues(
            ST_AddBand(
                ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0),
                1, '8BUI', 1, 0
            ),
            1, 2, 2, ARRAY[[9, 9], [9, 9]]::double precision[][]
        )
    ) AS poly
```

```
) foo
ORDER BY 1, 2;

 x | y | val
---+---+-----
 1 | 1 |   1
 1 | 2 |   1
 1 | 3 |   1
 2 | 1 |   1
 2 | 2 |   9
 2 | 3 |   9
 3 | 1 |   1
 3 | 2 |   9
 3 | 3 |   9
```

```
/*
The ST_SetValues() does the following...

+ - + - + - +          + - + - + - +
| 1 | 1 | 1 |          | 9 | 9 | 9 |
+ - + - + - +          + - + - + - +
| 1 | 1 | 1 |    =
>    | 9 |   | 9 |
+ - + - + - +          + - + - + - +
| 1 | 1 | 1 |          | 9 | 9 | 9 |
+ - + - + - +          + - + - + - +
*/
SELECT
    (poly).x,
    (poly).y,
    (poly).val
FROM (
SELECT
    ST_PixelAsPolygons(
        ST_SetValues(
            ST_AddBand(
                ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0),
                1, '8BUI', 1, 0
            ),
            1, 1, 1, ARRAY[[9, 9, 9], [9, NULL, 9], [9, 9, 9]]::double precision[][]
        )
    ) AS poly
) foo
ORDER BY 1, 2;

 x | y | val
---+---+-----
 1 | 1 |   9
 1 | 2 |   9
 1 | 3 |   9
 2 | 1 |   9
 2 | 2 |
 2 | 3 |   9
 3 | 1 |   9
 3 | 2 |   9
 3 | 3 |   9
```

```
/*
The ST_SetValues() does the following...

+ - + - + - +          + - + - + - +
```

```
| 1 | 1 | 1 |          | 9 | 9 | 9 |
+ - + - + - +          + - + - + - +
| 1 | 1 | 1 |    =
>    | 1 |   | 9 |
+ - + - + - +          + - + - + - +
| 1 | 1 | 1 |          | 9 | 9 | 9 |
+ - + - + - +          + - + - + - +
*/
SELECT
    (poly).x,
    (poly).y,
    (poly).val
FROM (
SELECT
    ST_PixelAsPolygons(
        ST_SetValues(
            ST_AddBand(
                ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0),
                1, '8BUI', 1, 0
            ),
            1, 1, 1,
                ARRAY[[9, 9, 9], [9, NULL, 9], [9, 9, 9]]::double precision[][],
                ARRAY[[false], [true]]::boolean[][]
        )
    ) AS poly
) foo
ORDER BY 1, 2;

 x | y | val
---+---+-----
 1 | 1 |   9
 1 | 2 |   1
 1 | 3 |   9
 2 | 1 |   9
 2 | 2 |
 2 | 3 |   9
 3 | 1 |   9
 3 | 2 |   9
 3 | 3 |   9
```

```
/*
The ST_SetValues() does the following...

+ - + - + - +          + - + - + - +
|   | 1 | 1 |          |   | 9 | 9 |
+ - + - + - +          + - + - + - +
| 1 | 1 | 1 |    =
>    | 1 |   | 9 |
+ - + - + - +          + - + - + - +
| 1 | 1 | 1 |          | 9 | 9 | 9 |
+ - + - + - +          + - + - + - +
*/
SELECT
    (poly).x,
    (poly).y,
    (poly).val
FROM (
SELECT
    ST_PixelAsPolygons(
        ST_SetValues(
            ST_SetValue(
                ST_AddBand(
```

```
                    ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0),
                    1, '8BUI', 1, 0
                ),
                1, 1, 1, NULL
            ),
            1, 1, 1,
                ARRAY[[9, 9, 9], [9, NULL, 9], [9, 9, 9]]::double precision[][],
                ARRAY[[false], [true]]::boolean[][],
                TRUE
        )
    ) AS poly
) foo
ORDER BY 1, 2;

 x | y | val
---+---+-----
 1 | 1 |
 1 | 2 |   1
 1 | 3 |   9
 2 | 1 |   9
 2 | 2 |
 2 | 3 |   9
 3 | 1 |   9
 3 | 2 |   9
 3 | 3 |   9
```

Exemplos: Variant 2

```
/*
The ST_SetValues() does the following...

+ - + - + - +          + - + - + - +
| 1 | 1 | 1 |          | 1 | 1 | 1 |
+ - + - + - +          + - + - + - +
| 1 | 1 | 1 |    =
>    | 1 | 9 | 9 |
+ - + - + - +          + - + - + - +
| 1 | 1 | 1 |          | 1 | 9 | 9 |
+ - + - + - +          + - + - + - +
*/
SELECT
    (poly).x,
    (poly).y,
    (poly).val
FROM (
SELECT
    ST_PixelAsPolygons(
        ST_SetValues(
            ST_AddBand(
                ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0),
                1, '8BUI', 1, 0
            ),
            1, 1, 1, ARRAY[[-1, -1, -1], [-1, 9, 9], [-1, 9, 9]]::double precision[][], -1
        )
    ) AS poly
) foo
ORDER BY 1, 2;

 x | y | val
---+---+-----
```

```
 1 | 1 |   1
 1 | 2 |   1
 1 | 3 |   1
 2 | 1 |   1
 2 | 2 |   9
 2 | 3 |   9
 3 | 1 |   1
 3 | 2 |   9
 3 | 3 |   9
```

```
/*
This example is like the previous one.  Instead of nosetvalue = -1, nosetvalue = NULL

The ST_SetValues() does the following...

+ - + - + - +          + - + - + - +
| 1 | 1 | 1 |          | 1 | 1 | 1 |
+ - + - + - +          + - + - + - +
| 1 | 1 | 1 |    =
>    | 1 | 9 | 9 |
+ - + - + - +          + - + - + - +
| 1 | 1 | 1 |          | 1 | 9 | 9 |
+ - + - + - +          + - + - + - +
*/
SELECT
    (poly).x,
    (poly).y,
    (poly).val
FROM (
SELECT
    ST_PixelAsPolygons(
        ST_SetValues(
            ST_AddBand(
                ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0),
                1, '8BUI', 1, 0
            ),
            1, 1, 1, ARRAY[[NULL, NULL, NULL], [NULL, 9, 9], [NULL, 9, 9]]::double ↩
                precision[][], NULL::double precision
        )
    ) AS poly
) foo
ORDER BY 1, 2;

 x | y | val
---+---+-----
 1 | 1 |   1
 1 | 2 |   1
 1 | 3 |   1
 2 | 1 |   1
 2 | 2 |   9
 2 | 3 |   9
 3 | 1 |   1
 3 | 2 |   9
 3 | 3 |   9
```

Exemplos: Variante 3

```
/*
The ST_SetValues() does the following...
```

```
+ - + - + - +          + - + - + - +
| 1 | 1 | 1 |          | 1 | 1 | 1 |
+ - + - + - +          + - + - + - +
| 1 | 1 | 1 |    =
>    | 1 | 9 | 9 |
+ - + - + - +          + - + - + - +
| 1 | 1 | 1 |          | 1 | 9 | 9 |
+ - + - + - +          + - + - + - +
*/
SELECT
    (poly).x,
    (poly).y,
    (poly).val
FROM (
SELECT
    ST_PixelAsPolygons(
        ST_SetValues(
            ST_AddBand(
                ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0),
                1, '8BUI', 1, 0
            ),
            1, 2, 2, 2, 2, 9
        )
    ) AS poly
) foo
ORDER BY 1, 2;

 x | y | val
---+---+-----
 1 | 1 |   1
 1 | 2 |   1
 1 | 3 |   1
 2 | 1 |   1
 2 | 2 |   9
 2 | 3 |   9
 3 | 1 |   1
 3 | 2 |   9
 3 | 3 |   9
```

```
/*
The ST_SetValues() does the following...

+ - + - + - +          + - + - + - +
| 1 | 1 | 1 |          | 1 | 1 | 1 |
+ - + - + - +          + - + - + - +
| 1 |   | 1 |    =
>    | 1 |   | 9 |
+ - + - + - +          + - + - + - +
| 1 | 1 | 1 |          | 1 | 9 | 9 |
+ - + - + - +          + - + - + - +
*/
SELECT
    (poly).x,
    (poly).y,
    (poly).val
FROM (
SELECT
    ST_PixelAsPolygons(
        ST_SetValues(
            ST_SetValue(
                ST_AddBand(
                    ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0),
```

```
                    1, '8BUI', 1, 0
                ),
                1, 2, 2, NULL
            ),
            1, 2, 2, 2, 2, 9, TRUE
        )
    ) AS poly
) foo
ORDER BY 1, 2;

 x | y | val
---+---+-----
 1 | 1 |   1
 1 | 2 |   1
 1 | 3 |   1
 2 | 1 |   1
 2 | 2 |
 2 | 3 |   9
 3 | 1 |   1
 3 | 2 |   9
 3 | 3 |   9
```

Exemplos: Variante 5

```
WITH foo AS (
    SELECT 1 AS rid, ST_AddBand(ST_MakeEmptyRaster(5, 5, 0, 0, 1, -1, 0, 0, 0), 1, '8BUI',  ↩
        0, 0) AS rast
), bar AS (
    SELECT 1 AS gid, 'SRID=0;POINT(2.5 -2.5)'::geometry geom UNION ALL
    SELECT 2 AS gid, 'SRID=0;POLYGON((1 -1, 4 -1, 4 -4, 1 -4, 1 -1))'::geometry geom UNION  ↩
        ALL
    SELECT 3 AS gid, 'SRID=0;POLYGON((0 0, 5 0, 5 -1, 1 -1, 1 -4, 0 -4, 0 0))'::geometry  ↩
        geom UNION ALL
    SELECT 4 AS gid, 'SRID=0;MULTIPOINT(0 0, 4 4, 4 -4)'::geometry
)
SELECT
    rid, gid, ST_DumpValues(ST_SetValue(rast, 1, geom, gid))
FROM foo t1
CROSS JOIN bar t2
ORDER BY rid, gid;

 rid | gid |                                                                                  st_dumpvalues
-----+-----+-------------------------------------------------------------------------------------------------------------------------

   1 |   1 | (1,"{{NULL,NULL,NULL,NULL,NULL},{NULL,NULL,NULL,NULL,NULL},{NULL,NULL,1,NULL,  ↩
       NULL},{NULL,NULL,NULL,NULL,NULL},{NULL,NULL,NULL,NULL,NULL}}")
   1 |   2 | (1,"{{NULL,NULL,NULL,NULL,NULL},{NULL,2,2,2,NULL},{NULL,2,2,2,NULL},{NULL ↩
       ,2,2,2,NULL},{NULL,NULL,NULL,NULL,NULL}}")
   1 |   3 | (1,"{{3,3,3,3,3},{3,NULL,NULL,NULL,NULL},{3,NULL,NULL,NULL,NULL},{3,NULL,NULL,  ↩
       NULL,NULL},{NULL,NULL,NULL,NULL,NULL}}")
   1 |   4 | (1,"{{4,NULL,NULL,NULL,NULL},{NULL,NULL,NULL,NULL,NULL},{NULL,NULL,NULL,NULL,  ↩
       NULL},{NULL,NULL,NULL,NULL,NULL},{NULL,NULL,NULL,NULL,4}}")
(4 rows)
```

A seguir está demonstrado que geomvals podem, mais tarde, sobrescrever no arranjo geomvals anteriores

```
WITH foo AS (
    SELECT 1 AS rid, ST_AddBand(ST_MakeEmptyRaster(5, 5, 0, 0, 1, -1, 0, 0, 0), 1, '8BUI',  ↩
        0, 0) AS rast
), bar AS (
```

```
    SELECT 1 AS gid, 'SRID=0;POINT(2.5 -2.5)'::geometry geom UNION ALL
    SELECT 2 AS gid, 'SRID=0;POLYGON((1 -1, 4 -1, 4 -4, 1 -4, 1 -1))'::geometry geom UNION  ↩
        ALL
    SELECT 3 AS gid, 'SRID=0;POLYGON((0 0, 5 0, 5 -1, 1 -1, 1 -4, 0 -4, 0 0))'::geometry  ↩
        geom UNION ALL
    SELECT 4 AS gid, 'SRID=0;MULTIPOINT(0 0, 4 4, 4 -4)'::geometry
)
SELECT
    t1.rid, t2.gid, t3.gid, ST_DumpValues(ST_SetValues(rast, 1, ARRAY[ROW(t2.geom, t2.gid), ↩
         ROW(t3.geom, t3.gid)]::geomval[]))
FROM foo t1
CROSS JOIN bar t2
CROSS JOIN bar t3
WHERE t2.gid = 1
    AND t3.gid = 2
ORDER BY t1.rid, t2.gid, t3.gid;

 rid | gid | gid |                                                                    st_dumpvalues
-----+-----+-----+-----------------------------------------------------------------------------------------------

   1 |   1 |   2 | (1,"{{NULL,NULL,NULL,NULL,NULL},{NULL,2,2,2,NULL},{NULL,2,2,2,NULL},{ ↩
      NULL,2,2,2,NULL},{NULL,NULL,NULL,NULL,NULL}}")
(1 row)
```

Este exemplo é o oposto do exemplo anterior

```
WITH foo AS (
    SELECT 1 AS rid, ST_AddBand(ST_MakeEmptyRaster(5, 5, 0, 0, 1, -1, 0, 0, 0), 1, '8BUI',  ↩
        0, 0) AS rast
), bar AS (
    SELECT 1 AS gid, 'SRID=0;POINT(2.5 -2.5)'::geometry geom UNION ALL
    SELECT 2 AS gid, 'SRID=0;POLYGON((1 -1, 4 -1, 4 -4, 1 -4, 1 -1))'::geometry geom UNION  ↩
        ALL
    SELECT 3 AS gid, 'SRID=0;POLYGON((0 0, 5 0, 5 -1, 1 -1, 1 -4, 0 -4, 0 0))'::geometry  ↩
        geom UNION ALL
    SELECT 4 AS gid, 'SRID=0;MULTIPOINT(0 0, 4 4, 4 -4)'::geometry
)
SELECT
    t1.rid, t2.gid, t3.gid, ST_DumpValues(ST_SetValues(rast, 1, ARRAY[ROW(t2.geom, t2.gid), ↩
         ROW(t3.geom, t3.gid)]::geomval[]))
FROM foo t1
CROSS JOIN bar t2
CROSS JOIN bar t3
WHERE t2.gid = 2
    AND t3.gid = 1
ORDER BY t1.rid, t2.gid, t3.gid;

 rid | gid | gid |                                                                    st_dumpvalues
-----+-----+-----+-----------------------------------------------------------------------------------------------

   1 |   2 |   1 | (1,"{{NULL,NULL,NULL,NULL,NULL},{NULL,2,2,2,NULL},{NULL,2,1,2,NULL},{ ↩
      NULL,2,2,2,NULL},{NULL,NULL,NULL,NULL,NULL}}")
(1 row)
```

Veja também

ST_Value, ST_SetValue, ST_PixelAsPolygons

10.6.14 ST_DumpValues

ST_DumpValues — Obtenha os valores da banda específica como um arranjo 2-dimensional.

Synopsis

setof record **ST_DumpValues**(raster rast , integer[] nband=NULL , boolean exclude_nodata_value=true);
double precision[][] **ST_DumpValues**(raster rast , integer nband , boolean exclude_nodata_value=true);

Descrição

Obtenha os valores da banda especificada como um arranjo 2-dimensional (o primeiro índice é linha, o segundo é coluna). Se `nband` for NULO ou não for fornecido, todas as bandas raster serão processadas.

Disponibilidade: 2.1.0

Exemplos

```
WITH foo AS (
    SELECT ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0), ↩
        1, '8BUI'::text, 1, 0), 2, '32BF'::text, 3, -9999), 3, '16BSI', 0, 0) AS rast
)
SELECT
    (ST_DumpValues(rast)).*
FROM foo;

 nband |                        valarray
-------+-----------------------------------------------------
     1 | {{1,1,1},{1,1,1},{1,1,1}}
     2 | {{3,3,3},{3,3,3},{3,3,3}}
     3 | {{NULL,NULL,NULL},{NULL,NULL,NULL},{NULL,NULL,NULL}}
(3 rows)
```

```
WITH foo AS (
    SELECT ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0), ↩
        1, '8BUI'::text, 1, 0), 2, '32BF'::text, 3, -9999), 3, '16BSI', 0, 0) AS rast
)
SELECT
    (ST_DumpValues(rast, ARRAY[3, 1])).*
FROM foo;

 nband |                        valarray
-------+-----------------------------------------------------
     3 | {{NULL,NULL,NULL},{NULL,NULL,NULL},{NULL,NULL,NULL}}
     1 | {{1,1,1},{1,1,1},{1,1,1}}
(2 rows)
```

```
WITH foo AS (
    SELECT ST_SetValue(ST_AddBand(ST_MakeEmptyRaster(3, 3, 0, 0, 1, -1, 0, 0, 0), 1, '8BUI ↩
        ', 1, 0), 1, 2, 5) AS rast
)
SELECT
    (ST_DumpValues(rast, 1))[2][1]
FROM foo;

 st_dumpvalues
---------------
             5
(1 row)
```

Veja também

ST_Value, ST_SetValue, ST_SetValues

10.6.15 ST_PixelOfValue

ST_PixelOfValue — Obtenha as coordenadas colunax, linhay do pixel cujos valores são iguais ao valor de pesquisa.

Synopsis

setof record **ST_PixelOfValue**(raster rast , integer nband , double precision[] search , boolean exclude_nodata_value=true);
setof record **ST_PixelOfValue**(raster rast , double precision[] search , boolean exclude_nodata_value=true);
setof record **ST_PixelOfValue**(raster rast , integer nband , double precision search , boolean exclude_nodata_value=true);
setof record **ST_PixelOfValue**(raster rast , double precision search , boolean exclude_nodata_value=true);

Descrição

Obtenha as coordenadas colunax, linhay do pixel cujos valores são iguais ao valor de pesquisa. Se nenhuma banda for especificada, então a banda 1 é assumida.

Disponibilidade: 2.1.0

Exemplos

```
SELECT
    (pixels).*
FROM (
    SELECT
        ST_PixelOfValue(
            ST_SetValue(
                ST_SetValue(
                    ST_SetValue(
                        ST_SetValue(
                            ST_SetValue(
                                ST_AddBand(
                                    ST_MakeEmptyRaster(5, 5, -2, 2, 1, -1, 0, 0, 0),
                                    '8BUI'::text, 1, 0
                                ),
                                1, 1, 0
                            ),
                            2, 3, 0
                        ),
                        3, 5, 0
                    ),
                    4, 2, 0
                ),
                5, 4, 255
            )
        , 1, ARRAY[1, 255]) AS pixels
) AS foo

 val | x | y
-----+---+---
   1 | 1 | 2
   1 | 1 | 3
   1 | 1 | 4
   1 | 1 | 5
```

```
  1 | 2 | 1
  1 | 2 | 2
  1 | 2 | 4
  1 | 2 | 5
  1 | 3 | 1
  1 | 3 | 2
  1 | 3 | 3
  1 | 3 | 4
  1 | 4 | 1
  1 | 4 | 3
  1 | 4 | 4
  1 | 4 | 5
  1 | 5 | 1
  1 | 5 | 2
  1 | 5 | 3
255 | 5 | 4
  1 | 5 | 5
```

10.7 Editores Raster

10.7.1 ST_SetGeoReference

ST_SetGeoReference — Coloque os parâmetros Georeference 6 em uma única chamada. Os números deverão ser separadospor espaço branco. Aceita entrar no formato GDAL ou ESRI. O padrão é GDAL.

Synopsis

raster **ST_SetGeoReference**(raster rast, text georefcoords, text format=GDAL);
raster **ST_SetGeoReference**(raster rast, double precision upperleftx, double precision upperlefty, double precision scalex, double precision scaley, double precision skewx, double precision skewy);

Descrição

Coloca os parâmetros georreferência 6 em uma única chamada. Aceita entrar no formato GDAL ou ESRI. O padrão é GDAL. Se 6 coordenadas não forem fornecidas, retornará null.

A diferença entre representações de formatos é a seguinte:

GDAL:

```
scalex skewy skewx scaley upperleftx upperlefty
```

ESRI:

```
scalex skewy skewx scaley upperleftx + scalex*0.5 upperlefty + scaley*0.5
```

Note
Se o raster tiver bandas fora do banco de dados, alterar a georreferência pode resultar em acesso incorreto dos dados de banda armazenados externamente.

Melhorias: 2.1.0 Adição da variante ST_SetGeoReference(raster, double precision, ...)

Exemplos

```
WITH foo AS (
    SELECT ST_MakeEmptyRaster(5, 5, 0, 0, 1, -1, 0, 0, 0) AS rast
)
SELECT
    0 AS rid, (ST_Metadata(rast)).*
FROM foo
UNION ALL
SELECT
    1, (ST_Metadata(ST_SetGeoReference(rast, '10 0 0 -10 0.1 0.1', 'GDAL'))).*
FROM foo
UNION ALL
SELECT
    2, (ST_Metadata(ST_SetGeoReference(rast, '10 0 0 -10 5.1 -4.9', 'ESRI'))).*
FROM foo
UNION ALL
SELECT
    3, (ST_Metadata(ST_SetGeoReference(rast, 1, 1, 10, -10, 0.001, 0.001))).*
FROM foo

 rid |     upperleftx     |     upperlefty     | width | height | scalex | scaley | skewx | ↩
     skewy | srid | numbands
-----+--------------------+--------------------+-------+--------+--------+--------+-------+-------+-
   0 |                  0 |                  0 |     5 |      5 |      1 |     -1 |     0 | ↩
           0 |    0 |        0
   1 |                0.1 |                0.1 |     5 |      5 |     10 |    -10 |     0 | ↩
           0 |    0 |        0
   2 | 0.0999999999999996 | 0.0999999999999996 |     5 |      5 |     10 |    -10 |     0 | ↩
           0 |    0 |        0
   3 |                  1 |                  1 |     5 |      5 |     10 |    -10 | 0.001 | ↩
       0.001 |    0 |        0
```

Veja também

ST_GeoReference, ST_ScaleX, ST_ScaleY, ST_UpperLeftX, ST_UpperLeftY

10.7.2 ST_SetRotation

ST_SetRotation — Põe a rotação do raster em radianos.

Synopsis

float8 **ST_SetRotation**(raster rast, float8 rotation);

Descrição

Gira o raster uniformemente. A rotação é em radianos. Recorra a World File para mais detalhes.

Exemplos

```
SELECT
  ST_ScaleX(rast1), ST_ScaleY(rast1), ST_SkewX(rast1), ST_SkewY(rast1),
  ST_ScaleX(rast2), ST_ScaleY(rast2), ST_SkewX(rast2), ST_SkewY(rast2)
```

```
FROM (
  SELECT ST_SetRotation(rast, 15) AS rast1, rast as rast2 FROM dummy_rast
) AS foo;
      st_scalex       |      st_scaley       |      st_skewx       |      st_skewy       |  ↩
          st_scalex | st_scaley | st_skewx | st_skewy
--------------------+----------------------+---------------------+---------------------+----------+--

   -1.51937582571764 |   -2.27906373857646 |   1.95086352047135 |   1.30057568031423 |  ↩
               2 |          3 |         0 |          0
 -0.0379843956429411 | -0.0379843956429411 | 0.0325143920078558 | 0.0325143920078558 |  ↩
          0.05 |      -0.05 |         0 |          0
```

Veja também

ST_Rotation, ST_ScaleX, ST_ScaleY, ST_SkewX, ST_SkewY

10.7.3 ST_SetScale

ST_SetScale — Coloca os tamanhos X e Y dos pixeis em unidades do sistema referencial de coordenadas. Número unidades/pixel largura/altura.

Synopsis

raster **ST_SetScale**(raster rast, float8 xy);
raster **ST_SetScale**(raster rast, float8 x, float8 y);

Descrição

Coloca os tamanhos X e Y dos pixeis em unidades do sistema referencial de coordenadas. Número unidades/pixel largura/altura. Se apenas uma unidade passar, o X e Y assumido são o mesmo número.

Note
ST_SetScale é diferente de ST_Rescale onde a ST_SetScale não resample o raster para combinar com a extensão. Apenas altera os metadados (ou georreferência) do raster, para corrigir uma escala originalmente mal especificada. A ST_SetScale resulta em um raster tendo largura e altura diferentes, calculadas para caber na extensão geográfica do raster de entrada. A ST_SetScale não modifica a largura, nem a altura do raster.

Alterações: 2.0.0 Nas versões WKTRaster era chamado de ST_SetPixelSize. Foi modificado na 2.0.0.

Exemplos

```
UPDATE dummy_rast
    SET rast = ST_SetScale(rast, 1.5)
WHERE rid = 2;

SELECT ST_ScaleX(rast) As pixx, ST_ScaleY(rast) As pixy, Box3D(rast) As newbox
FROM dummy_rast
WHERE rid = 2;

 pixx | pixy |                    newbox
------+------+-----------------------------------------------
  1.5 |  1.5 | BOX(3427927.75 5793244 0, 3427935.25 5793251.5 0)
```

```
UPDATE dummy_rast
    SET rast = ST_SetScale(rast, 1.5, 0.55)
WHERE rid = 2;

SELECT ST_ScaleX(rast) As pixx, ST_ScaleY(rast) As pixy, Box3D(rast) As newbox
FROM dummy_rast
WHERE rid = 2;

 pixx | pixy |                   newbox
------+------+-------------------------------------------
  1.5 | 0.55 | BOX(3427927.75 5793244 0,3427935.25 5793247 0)
```

Veja também

ST_ScaleX, ST_ScaleY, Caixa3D

10.7.4 ST_SetSkew

ST_SetSkew — Coloca as georreferências X e Y distorcidas (ou parâmetro de rotação). Se somente um passar, coloca o X e o Y no mesmo valor.

Synopsis

raster **ST_SetSkew**(raster rast, float8 skewxy);
raster **ST_SetSkew**(raster rast, float8 skewx, float8 skewy);

Descrição

Coloca as georreferências X e Y distorcidas (ou parâmetro de rotação). Se somente um passar, coloca o X e o Y no mesmo valor. Recorra a World File para mais detalhes.

Exemplos

```
-- Example 1
UPDATE dummy_rast SET rast = ST_SetSkew(rast,1,2) WHERE rid = 1;
SELECT rid, ST_SkewX(rast) As skewx, ST_SkewY(rast) As skewy,
    ST_GeoReference(rast) as georef
FROM dummy_rast WHERE rid = 1;

rid | skewx | skewy |    georef
----+-------+-------+--------------
  1 |     1 |     2 | 2.0000000000
                    : 2.0000000000
                    : 1.0000000000
                    : 3.0000000000
                    : 0.5000000000
                    : 0.5000000000
```

```
-- Example 2 set both to same number:
UPDATE dummy_rast SET rast = ST_SetSkew(rast,0) WHERE rid = 1;
SELECT rid, ST_SkewX(rast) As skewx, ST_SkewY(rast) As skewy,
    ST_GeoReference(rast) as georef
FROM dummy_rast WHERE rid = 1;
```

```
 rid | skewx | skewy |    georef
-----+-------+-------+--------------
   1 |     0 |     0 | 2.0000000000
                     : 0.0000000000
                     : 0.0000000000
                     : 3.0000000000
                     : 0.5000000000
                     : 0.5000000000
```

Veja também

ST_GeoReference, ST_SetGeoReference, ST_SkewX, ST_SkewY

10.7.5 ST_SetSRID

ST_SetSRID — Coloca o SRID de um raster em um srid inteiro específico definido na tabela spatial_ref_sys.

Synopsis

raster **ST_SetSRID**(raster rast, integer srid);

Descrição

Coloca o SRID em um raster para um valor inteiro específico.

Note
Esta função não transforma o raster em forma alguma - simplesmente coloca metadados definindo a referência espacial do sistema de coordenadas referência que está sendo usado. É útil para futuras transformações.

Veja também

Section 4.5, ST_SRID

10.7.6 ST_SetUpperLeft

ST_SetUpperLeft — Sets the value of the upper left corner of the pixel of the raster to projected X and Y coordinates.

Synopsis

raster **ST_SetUpperLeft**(raster rast, double precision x, double precision y);

Descrição

Set the value of the upper left corner of raster to the projected X and Y coordinates

Exemplos

```
SELECT ST_SetUpperLeft(rast,-71.01,42.37)
FROM dummy_rast
WHERE rid = 2;
```

Veja também

ST_UpperLeftX, ST_UpperLeftY

10.7.7 ST_Resample

ST_Resample — Resample um raster usando um algorítimo específico, novas dimensões, um canto aleatório da grade e um conjunto de rasters georreferenciando atributos definidos ou emprestados de outro raster.

Synopsis

raster **ST_Resample**(raster rast, integer width, integer height, double precision gridx=NULL, double precision gridy=NULL, double precision skewx=0, double precision skewy=0, text algorithm=NearestNeighbor, double precision maxerr=0.125);
raster **ST_Resample**(raster rast, double precision scalex=0, double precision scaley=0, double precision gridx=NULL, double precision gridy=NULL, double precision skewx=0, double precision skewy=0, text algorithm=NearestNeighbor, double precision maxerr=0.125);
raster **ST_Resample**(raster rast, raster ref, text algorithm=NearestNeighbor, double precision maxerr=0.125, boolean usescale=true);
raster **ST_Resample**(raster rast, raster ref, boolean usescale, text algorithm=NearestNeighbor, double precision maxerr=0.125);

Descrição

Resample um raster usando um algorítimo específico, novas dimensões (largura & altura), um canto de grade (gradex & gradey) e um conjunto de rasters georreferenciando atributos (scalex, scaley, skewx & skewy) definidos ou emprestados de outro raster. Se estiver utilizando uma referência raster, os dois rasters devem possuir o mesmo SRID.

New pixel values are computed using one of the following resampling algorithms:

- NearestNeighbor (english or american spelling)
- Bilinear
- Cubic
- CubicSpline
- Lanczos
- Max
- Min

The default is NearestNeighbor which is the fastest but results in the worst interpolation.

Uma porcentagem maxerror de 0.125 é usada se nenhum `maxerr` for especificado.

Note
Recorra a: GDAL Warp resampling methods para mais detalhes.

Disponibilidade: 2.0.0 Requer GDAL 1.6.1+

Enhanced: 3.4.0 max and min resampling options added

Exemplos

```
SELECT
    ST_Width(orig) AS orig_width,
    ST_Width(reduce_100) AS new_width
FROM (
    SELECT
        rast AS orig,
        ST_Resample(rast,100,100) AS reduce_100
    FROM aerials.boston
    WHERE ST_Intersects(rast,
        ST_Transform(
            ST_MakeEnvelope(-71.128, 42.2392,-71.1277, 42.2397, 4326),26986)
    )
    LIMIT 1
) AS foo;

 orig_width | new_width
------------+-------------
        200 |         100
```

Veja também

ST_Rescale, ST_Resize, ST_Transform

10.7.8 ST_Rescale

ST_Rescale — Resample a raster by adjusting only its scale (or pixel size). New pixel values are computed using the NearestNeighbor (english or american spelling), Bilinear, Cubic, CubicSpline, Lanczos, Max or Min resampling algorithm. Default is NearestNeighbor.

Synopsis

raster **ST_Rescale**(raster rast, double precision scalexy, text algorithm=NearestNeighbor, double precision maxerr=0.125);
raster **ST_Rescale**(raster rast, double precision scalex, double precision scaley, text algorithm=NearestNeighbor, double precision maxerr=0.125);

Descrição

Resample a raster by adjusting only its scale (or pixel size). New pixel values are computed using one of the following resampling algorithms:

- NearestNeighbor (english or american spelling)
- Bilinear
- Cubic
- CubicSpline
- Lanczos
- Max
- Min

The default is NearestNeighbor which is the fastest but results in the worst interpolation.

`scalex` e `scaley` definem o tamanho do pixel. A escalay deve ser, geralmente, negativa para ser um raster bem orientado.

Quando a nova escalax ou escalay não é divisora da largura ou altura do raster, a extensão do raster resultante é expandido para encerrar a extensão do raster fornecido. Se quiser certificar-se de reter a entrada exata, veja ST_Resize

`maxerr` is the threshold for transformation approximation by the resampling algorithm (in pixel units). A default of 0.125 is used if no `maxerr` is specified, which is the same value used in GDAL gdalwarp utility. If set to zero, no approximation takes place.

Note
Recorra a: GDAL Warp resampling methods para mais detalhes.

Note
ST_Rescale é diferente de ST_SetScale onde a ST_SetScale não resample o raster para combinar com a extensão. A ST_SetScale apenas altera os metadados (ou georreferência) do raster, para corrigir uma escala originalmente mal especificada. A ST_Rescale resulta em um raster tendo largura e altura diferentes, calculadas para caber na extensão geográfica do raster de entrada. A ST_SetScale não modifica a largura, nem a altura do raster.

Disponibilidade: 2.0.0 Requer GDAL 1.6.1+

Enhanced: 3.4.0 max and min resampling options added

Alterações: 2.1.0 Funciona em rasters sem SRID

Exemplos

Um exemplo simples de reescalar um raster de um tamanho de pixel de 0.001 grau para um pixel de tamanho 0.0015 grau.

```
-- the original raster pixel size
SELECT ST_PixelWidth(ST_AddBand(ST_MakeEmptyRaster(100, 100, 0, 0, 0.001, -0.001, 0, 0,  ↩
    4269), '8BUI'::text, 1, 0)) width

   width
----------
0.001

-- the rescaled raster raster pixel size
SELECT ST_PixelWidth(ST_Rescale(ST_AddBand(ST_MakeEmptyRaster(100, 100, 0, 0, 0.001,  ↩
    -0.001, 0, 0, 4269), '8BUI'::text, 1, 0), 0.0015)) width

   width
----------
0.0015
```

Veja também

ST_Resize, ST_Resample, ST_SetScale, ST_ScaleX, ST_ScaleY, ST_Transform

10.7.9 ST_Reskew

ST_Reskew — Resample um raster ajustando somente sua inclinação (ou tamanho de pixel). Novos valores de pixel são calculados usando o algorítimo NearestNeighbor (english or american spelling), Bilinear, Cubic, CubicSpline ou Lanczos. O padrão é NearestNeighbor.

Synopsis

raster **ST_Reskew**(raster rast, double precision skewxy, text algorithm=NearestNeighbor, double precision maxerr=0.125);
raster **ST_Reskew**(raster rast, double precision skewx, double precision skewy, text algorithm=NearestNeighbor, double precision maxerr=0.125);

Descrição

Resample um raster ajustando somente sua inclinação (ou tamanho de pixel). Novos valores de pixel são calculados usando o algorítimo NearestNeighbor (english or american spelling), Bilinear, Cubic, CubicSpline ou Lanczos. O padrão é NearestNeighbor pois é o mais rápido, mas resulta na pior interpolação.

`skewx` e `skewy` definem a nova distorção.

A extensão do novo raster irá encerrar a extensão do raster fornecido.

Uma porcentagem maxerror de 0.125 se nenhum `maxerr` for especificado.

Note
Recorra a: GDAL Warp resampling methods para mais detalhes.

Note
ST_Reskew é diferente de ST_SetSkew onde a ST_SetSkew não resample o raster para combinar com a extensão. A ST_SetScale apenas altera os metadados (ou georreferência) do raster, para corrigir uma escala originalmente mal especificada. A ST_Reskew resulta em um raster tendo largura e altura diferentes, calculadas para caber na extensão geográfica do raster de entrada. A ST_SetSkew não modifica a largura, nem a altura do raster.

Disponibilidade: 2.0.0 Requer GDAL 1.6.1+

Alterações: 2.1.0 Funciona em rasters sem SRID

Exemplos

Um exemplo simples de reskewing um raster de uma inclinação de 0.0 para uma de 0.0015.

```
-- the original raster non-rotated
SELECT ST_Rotation(ST_AddBand(ST_MakeEmptyRaster(100, 100, 0, 0, 0.001, -0.001, 0, 0, 4269) ↩
    , '8BUI'::text, 1, 0));

-- result
0

-- the reskewed raster raster rotation
SELECT ST_Rotation(ST_Reskew(ST_AddBand(ST_MakeEmptyRaster(100, 100, 0, 0, 0.001, -0.001,  ↩
    0, 0, 4269), '8BUI'::text, 1, 0), 0.0015));

-- result
-0.982793723247329
```

Veja também

ST_Resample, ST_Rescale, ST_SetSkew, ST_SetRotation, ST_SkewX, ST_SkewY, ST_Transform

10.7.10 ST_SnapToGrid

ST_SnapToGrid — Resample um raster encaixando-o em uma grade. Novos valores de pixel são calculados usando o algorítimo NearestNeighbor (english or american spelling), Bilinear, Cubic, CubicSpline ou Lanczos. O padrão é NearestNeighbor.

Synopsis

raster **ST_SnapToGrid**(raster rast, double precision gridx, double precision gridy, text algorithm=NearestNeighbor, double precision maxerr=0.125, double precision scalex=DEFAULT 0, double precision scaley=DEFAULT 0);
raster **ST_SnapToGrid**(raster rast, double precision gridx, double precision gridy, double precision scalex, double precision scaley, text algorithm=NearestNeighbor, double precision maxerr=0.125);
raster **ST_SnapToGrid**(raster rast, double precision gridx, double precision gridy, double precision scalexy, text algorithm=NearestNeigh
double precision maxerr=0.125);

Descrição

Resample um raster encaixando-o em uma grade definida por um pixel de canto (gridx & gridy) e opcionalmente um tamanho de pixel (scalex & scaley). Novos valores de pixel são calculados usando o algorítimo NearestNeighbor (english or american spelling), Bilinear, Cubic, CubicSpline ou Lanczos. O padrão é NearestNeighbor pois é o mais rápido, mas resulta na pior interpolação.

`gridx` e `gridy` definem um canto pixel aleatório da nova grade. Não é necessário o canto esquerdo superior do novo raster e não precisa estar dentro do limite da extensão do novo raster.

You can optionally define the pixel size of the new grid with `scalex` and `scaley`.

A extensão do novo raster irá encerrar a extensão do raster fornecido.

Uma porcentagem maxerror de 0.125 se nenhum `maxerr` for especificado.

Note
Recorra a: GDAL Warp resampling methods para mais detalhes.

Note
Use ST_Resample se precisar de mais controle sobre os parâmetros da grade.

Disponibilidade: 2.0.0 Requer GDAL 1.6.1+

Alterações: 2.1.0 Funciona em rasters sem SRID

Exemplos

Um exemplo simples movendo um raster para um grade um pouco diferente.

```
-- the original raster upper left X
SELECT ST_UpperLeftX(ST_AddBand(ST_MakeEmptyRaster(10, 10, 0, 0, 0.001, -0.001, 0, 0, 4269) ↩
    , '8BUI'::text, 1, 0));
-- result
0

-- the upper left of raster after snapping
SELECT ST_UpperLeftX(ST_SnapToGrid(ST_AddBand(ST_MakeEmptyRaster(10, 10, 0, 0, 0.001,  ↩
    -0.001, 0, 0, 4269), '8BUI'::text, 1, 0), 0.0002, 0.0002));
```

```
--result
-0.0008
```

Veja também

ST_Resample, ST_Rescale, ST_UpperLeftX, ST_UpperLeftY

10.7.11 ST_Resize

ST_Resize — Redimensiona largura/altura novas para um raster

Synopsis

raster **ST_Resize**(raster rast, integer width, integer height, text algorithm=NearestNeighbor, double precision maxerr=0.125);
raster **ST_Resize**(raster rast, double precision percentwidth, double precision percentheight, text algorithm=NearestNeighbor, double precision maxerr=0.125);
raster **ST_Resize**(raster rast, text width, text height, text algorithm=NearestNeighbor, double precision maxerr=0.125);

Descrição

Redimensiona novas largura/altura para um raster. Elas podem ser especificadas no número exato de pixeis ou uma porcentagem delas. A extensão do novo raster será a mesma da extensão do raster fornecido.

Novos valores de pixel são calculados usando o algorítimo NearestNeighbor (english or american spelling), Bilinear, Cubic, CubicSpline ou Lanczos. O padrão é NearestNeighbor por ser mais rápido, porém resulta na pior interpolação.

A Variante 1 espera a largura/altura atual do raster de saída.

A Variante 2 espera os valores decimais entre zero (0) e um (1) indicando a porcentagem da largura/altura do raster.

A Variante 3 pega qualquer largura/altura do raster de saída ou uma porcentagem textual ("20%") indicando a porcentagem da largura/altura do raster de entrada.

Disponibilidade: 2.1.0 Requer GDAL 1.6.1+

Exemplos

```
WITH foo AS(
SELECT
    1 AS rid,
    ST_Resize(
        ST_AddBand(
            ST_MakeEmptyRaster(1000, 1000, 0, 0, 1, -1, 0, 0, 0)
            , 1, '8BUI', 255, 0
        )
    , '50%', '500') AS rast
UNION ALL
SELECT
    2 AS rid,
    ST_Resize(
        ST_AddBand(
            ST_MakeEmptyRaster(1000, 1000, 0, 0, 1, -1, 0, 0, 0)
            , 1, '8BUI', 255, 0
        )
    , 500, 100) AS rast
UNION ALL
```

```
SELECT
    3 AS rid,
    ST_Resize(
        ST_AddBand(
            ST_MakeEmptyRaster(1000, 1000, 0, 0, 1, -1, 0, 0, 0)
            , 1, '8BUI', 255, 0
        )
    , 0.25, 0.9) AS rast
), bar AS (
    SELECT rid, ST_Metadata(rast) AS meta, rast FROM foo
)
SELECT rid, (meta).* FROM bar

 rid | upperleftx | upperlefty | width | height | scalex | scaley | skewx | skewy | srid |  ↩
     numbands
-----+------------+------------+-------+--------+--------+--------+-------+-------+------+----------

   1 |          0 |          0 |   500 |    500 |      1 |     -1 |     0 |     0 |    0 |  ↩
             1
   2 |          0 |          0 |   500 |    100 |      1 |     -1 |     0 |     0 |    0 |  ↩
             1
   3 |          0 |          0 |   250 |    900 |      1 |     -1 |     0 |     0 |    0 |  ↩
             1
(3 rows)
```

Veja também

ST_Resample, ST_Rescale, ST_Reskew, ST_SnapToGrid

10.7.12 ST_Transform

ST_Transform — Reprojeta um raster em um sistema de referência espacial conhecido para outro usando um algorítimo resampling especificado. As opções são NearestNeighbor, Bilinear, Cubic, CubicSpline, Lanczos com o padrão sendo NearestNeighbor.

Synopsis

raster **ST_Transform**(raster rast, integer srid, text algorithm=NearestNeighbor, double precision maxerr=0.125, double precision scalex, double precision scaley);
raster **ST_Transform**(raster rast, integer srid, double precision scalex, double precision scaley, text algorithm=NearestNeighbor, double precision maxerr=0.125);
raster **ST_Transform**(raster rast, raster alignto, text algorithm=NearestNeighbor, double precision maxerr=0.125);

Descrição

Reprojeta um raster em um sistema de referência espacial conhecido para outro usando um algorítimo pixel warping especificado. Usa "NearestNeighbor" se nenhum algorítimo for especificado e a porcentagem maxerror de 0.125 se nenhum maxerr for especificado.

As opções de algorítimo são: 'NearestNeighbor', 'Bilinear', 'Cubic', 'CubicSpline', e 'Lanczos'. Recorra a: GDAL Warp resampling methods para mais detalhes.

Geralmente, aST_Transform confundida com a ST_SetSRID(). Na verdade, a ST_Transform modifica as coordenadas de um raster (e resample os valores do pixel) de um sistema de referência espacial para outro, enquanto a ST_SetSRID() só altera o identificador de SRID do raster.

Diferente das outras variantes, a 3 requer um raster referência como `alignto`. O raster transformado será alterado para o sistema de referência espacial (SRID) do raster referência e será alinhado (ST_SameAlignment = VERDADE) ao raster referência.

Note
Se achar que seu suporte de transformação não estiver funcionando corretamente, talvez precise colocar a variável de ambiente PROJSO na biblioteca de projeção .so ou .dll que seu PostGIS está usando. Isto só precisará ter o mesmo nome do arquivo, Então, por exemplo, no Windows, você iria em Painel de Controle -> Sitema -> Variáveis de Ambiente adicionar um sistema variável chamado `PROJSO` e colocar em `libproj.dll` (se estiver usando proj 4.6.1). Você terá que reiniciar seu serviço/daemon PostgreSQL depois dessa alteração.

Warning
When transforming a coverage of tiles, you almost always want to use a reference raster to insure same alignment and no gaps in your tiles as demonstrated in example: Variant 3.

Disponibilidade: 2.0.0 Requer GDAL 1.6.1+

Melhorias: 2.1.0 Adição da variante ST_Transform(rast, alignto)

Exemplos

```
SELECT ST_Width(mass_stm) As w_before, ST_Width(wgs_84) As w_after,
  ST_Height(mass_stm) As h_before, ST_Height(wgs_84) As h_after
    FROM
    ( SELECT rast As mass_stm, ST_Transform(rast,4326) As wgs_84
  ,  ST_Transform(rast,4326, 'Bilinear') AS wgs_84_bilin
       FROM aerials.o_2_boston
           WHERE ST_Intersects(rast,
               ST_Transform(ST_MakeEnvelope(-71.128, 42.2392,-71.1277, 42.2397, 4326) ↩
                   ,26986) )
       LIMIT 1) As foo;

 w_before | w_after | h_before | h_after
----------+---------+----------+---------
      200 |     228 |      200 |     170
```

original mass state plane meters (mass_stm)

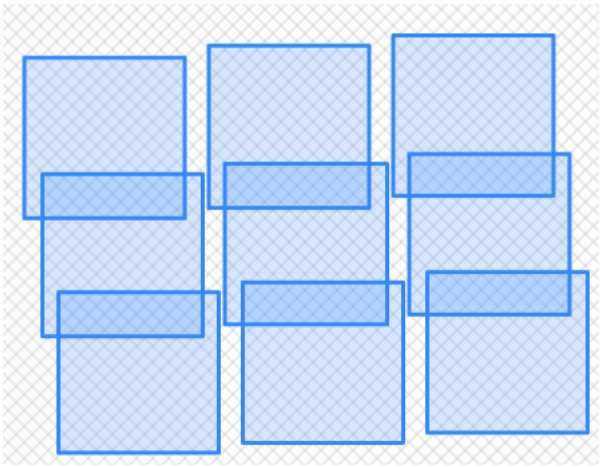
Depois transforma para wgs 84 long lat (wgs_84)

Depois transforma para wgs 84 long lat algorítimo bilinear em vez do padrão NN (wgs_84_bilin)

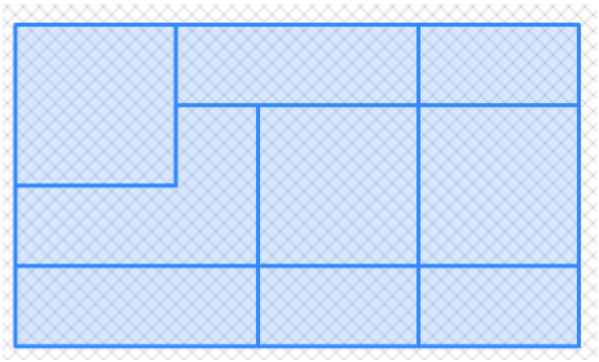
Exemplos: Variante 3

A seguir está a diferença entre usar ST_Transform(raster, srid) e ST_Transform(raster, alignto)

```
WITH foo AS (
    SELECT 0 AS rid, ST_AddBand(ST_MakeEmptyRaster(2, 2, -500000, 600000, 100, -100, 0, 0,  ↩
        2163), 1, '16BUI', 1, 0) AS rast UNION ALL
    SELECT 1, ST_AddBand(ST_MakeEmptyRaster(2, 2, -499800, 600000, 100, -100, 0, 0, 2163),  ↩
        1, '16BUI', 2, 0) AS rast UNION ALL
    SELECT 2, ST_AddBand(ST_MakeEmptyRaster(2, 2, -499600, 600000, 100, -100, 0, 0, 2163),  ↩
        1, '16BUI', 3, 0) AS rast UNION ALL

    SELECT 3, ST_AddBand(ST_MakeEmptyRaster(2, 2, -500000, 599800, 100, -100, 0, 0, 2163),  ↩
        1, '16BUI', 10, 0) AS rast UNION ALL
    SELECT 4, ST_AddBand(ST_MakeEmptyRaster(2, 2, -499800, 599800, 100, -100, 0, 0, 2163),  ↩
        1, '16BUI', 20, 0) AS rast UNION ALL
    SELECT 5, ST_AddBand(ST_MakeEmptyRaster(2, 2, -499600, 599800, 100, -100, 0, 0, 2163),  ↩
        1, '16BUI', 30, 0) AS rast UNION ALL

    SELECT 6, ST_AddBand(ST_MakeEmptyRaster(2, 2, -500000, 599600, 100, -100, 0, 0, 2163),  ↩
        1, '16BUI', 100, 0) AS rast UNION ALL
    SELECT 7, ST_AddBand(ST_MakeEmptyRaster(2, 2, -499800, 599600, 100, -100, 0, 0, 2163),  ↩
        1, '16BUI', 200, 0) AS rast UNION ALL
    SELECT 8, ST_AddBand(ST_MakeEmptyRaster(2, 2, -499600, 599600, 100, -100, 0, 0, 2163),  ↩
        1, '16BUI', 300, 0) AS rast
), bar AS (
    SELECT
        ST_Transform(rast, 4269) AS alignto
    FROM foo
    LIMIT 1
), baz AS (
    SELECT
        rid,
        rast,
        ST_Transform(rast, 4269) AS not_aligned,
        ST_Transform(rast, alignto) AS aligned
    FROM foo
    CROSS JOIN bar
)
SELECT
    ST_SameAlignment(rast) AS rast,
    ST_SameAlignment(not_aligned) AS not_aligned,
    ST_SameAlignment(aligned) AS aligned
FROM baz

 rast | not_aligned | aligned
------+-------------+---------
 t    | f           | t
```



not_aligned



aligned

Veja também

ST_Transform, ST_SetSRID

10.8 Editores de Banda Raster

10.8.1 ST_SetBandNoDataValue

ST_SetBandNoDataValue — Coloca o valor da banda que não representa nenhum dado. A banda 1 é assumida se nenhuma banda for especificada. Para marcar uma banda como tendo nenhum valor nodata, coloca ele = NULL.

Synopsis

raster **ST_SetBandNoDataValue**(raster rast, double precision nodatavalue);
raster **ST_SetBandNoDataValue**(raster rast, integer band, double precision nodatavalue, boolean forcechecking=false);

Descrição

Coloca o valor que não representa nenhum dado para a banda. A banda 1 é assumida se não especificada. Isso irá afetar os resultados de ST_Polygon, ST_DumpAsPolygons, e as funções ST_PixelAs...().

Exemplos

```
-- change just first band no data value
UPDATE dummy_rast
    SET rast = ST_SetBandNoDataValue(rast,1, 254)
WHERE rid = 2;

-- change no data band value of bands 1,2,3
UPDATE dummy_rast
    SET rast =
        ST_SetBandNoDataValue(
            ST_SetBandNoDataValue(
```

```
                ST_SetBandNoDataValue(
                    rast,1, 254)
                ,2,99),
                3,108)
        WHERE rid = 2;

-- wipe out the nodata value this will ensure all pixels are considered for all processing ↩
    functions
UPDATE dummy_rast
    SET rast = ST_SetBandNoDataValue(rast,1, NULL)
WHERE rid = 2;
```

Veja também

ST_BandNoDataValue, ST_NumBands

10.8.2 ST_SetBandIsNoData

ST_SetBandIsNoData — Coloca a bandeira isnodata da banda como VERDADE.

Synopsis

raster **ST_SetBandIsNoData**(raster rast, integer band=1);

Descrição

Coloca a bandeira isnodata para a banda como verdade. A banda 1 é assumida se não especificada. Esta função deveria ser chamada apenas quando a bandeira for considerada suja. Isto é, quando a chamada resultado ST_BandIsNoData for diferente usando VERDADEIRO como último argumento e sem usá-lo.

Disponibilidade: 2.0.0

Exemplos

```
-- Create dummy table with one raster column
create table dummy_rast (rid integer, rast raster);

-- Add raster with two bands, one pixel/band. In the first band, nodatavalue = pixel value ↩
    = 3.
-- In the second band, nodatavalue = 13, pixel value = 4
insert into dummy_rast values(1,
(
'01' -- little endian (uint8 ndr)
||
'0000' -- version (uint16 0)
||
'0200' -- nBands (uint16 0)
||
'17263529ED684A3F' -- scaleX (float64 0.000805965234044584)
||
'F9253529ED684ABF' -- scaleY (float64 -0.00080596523404458)
||
'1C9F33CE69E352C0' -- ipX (float64 -75.5533328537098)
||
'718F0E9A27A44840' -- ipY (float64 49.2824585505576)
||
```

```
'ED50EB853EC32B3F' -- skewX (float64 0.000211812383858707)
||
'7550EB853EC32B3F' -- skewY (float64 0.000211812383858704)
||
'E6100000' -- SRID (int32 4326)
||
'0100' -- width (uint16 1)
||
'0100' -- height (uint16 1)
||
'4' -- hasnodatavalue set to true, isnodata value set to false (when it should be true)
||
'2' -- first band type (4BUI)
||
'03' -- novalue==3
||
'03' -- pixel(0,0)==3 (same that nodata)
||
'0' -- hasnodatavalue set to false
||
'5' -- second band type (16BSI)
||
'0D00' -- novalue==13
||
'0400' -- pixel(0,0)==4
)::raster
);

select st_bandisnodata(rast, 1) from dummy_rast where rid = 1; -- Expected false
select st_bandisnodata(rast, 1, TRUE) from dummy_rast where rid = 1; -- Expected true

-- The isnodata flag is dirty. We are going to set it to true
update dummy_rast set rast = st_setbandisnodata(rast, 1) where rid = 1;


select st_bandisnodata(rast, 1) from dummy_rast where rid = 1; -- Expected true
```

Veja também

ST_BandNoDataValue, ST_NumBands, ST_SetBandNoDataValue, ST_BandIsNoData

10.8.3 ST_SetBandPath

ST_SetBandPath — Update the external path and band number of an out-db band

Synopsis

raster **ST_SetBandPath**(raster rast, integer band, text outdbpath, integer outdbindex, boolean force=false);

Descrição

Updates an out-db band's external raster file path and external band number.

Note
If `force` is set to true, no tests are done to ensure compatibility (e.g. alignment, pixel support) between the external raster file and the PostGIS raster. This mode is intended for file system changes where the external raster resides.

Note
Internally, this method replaces the PostGIS raster's band at index `band` with a new band instead of updating the existing path information.

Availability: 2.5.0

Exemplos

```
WITH foo AS (
    SELECT
        ST_AddBand(NULL::raster, '/home/pele/devel/geo/postgis-git/raster/test/regress/ ↩
            loader/Projected.tif', NULL::int[]) AS rast
)
SELECT
    1 AS query,
    *
FROM ST_BandMetadata(
    (SELECT rast FROM foo),
    ARRAY[1,3,2]::int[]
)
UNION ALL
SELECT
    2,
    *
FROM ST_BandMetadata(
    (
        SELECT
            ST_SetBandPath(
                rast,
                2,
                '/home/pele/devel/geo/postgis-git/raster/test/regress/loader/Projected2.tif ↩
                    ',
                1
            ) AS rast
        FROM foo
    ),
    ARRAY[1,3,2]::int[]
)
ORDER BY 1, 2;

 query | bandnum | pixeltype | nodatavalue | isoutdb |  ↩
                                        path                                         |  ↩
     outdbbandnum
-------+---------+-----------+-------------+---------+----------------------------------------------

     1 |       1 | 8BUI      |             | t       | /home/pele/devel/geo/postgis-git/ ↩
         raster/test/regress/loader/Projected.tif       |            1
     1 |       2 | 8BUI      |             | t       | /home/pele/devel/geo/postgis-git/ ↩
         raster/test/regress/loader/Projected.tif       |            2
     1 |       3 | 8BUI      |             | t       | /home/pele/devel/geo/postgis-git/ ↩
         raster/test/regress/loader/Projected.tif       |            3
```

```
 2 |       1 | 8BUI      |             | t       | /home/pele/devel/geo/postgis-git/ ↩
     raster/test/regress/loader/Projected.tif       |           1
 2 |       2 | 8BUI      |             | t       | /home/pele/devel/geo/postgis-git/ ↩
raster/test/regress/loader/Projected2.tif      |           1
 2 |       3 | 8BUI      |             | t       | /home/pele/devel/geo/postgis-git/ ↩
     raster/test/regress/loader/Projected.tif       |           3
```

Veja também

ST_BandMetaData, ST_SetBandIndex

10.8.4 ST_SetBandIndex

ST_SetBandIndex — Update the external band number of an out-db band

Synopsis

raster **ST_SetBandIndex**(raster rast, integer band, integer outdbindex, boolean force=false);

Descrição

Updates an out-db band's external band number. This does not touch the external raster file associated with the out-db band

Note
If `force` is set to true, no tests are done to ensure compatibility (e.g. alignment, pixel support) between the external raster file and the PostGIS raster. This mode is intended for where bands are moved around in the external raster file.

Note
Internally, this method replaces the PostGIS raster's band at index `band` with a new band instead of updating the existing path information.

Availability: 2.5.0

Exemplos

```
WITH foo AS (
    SELECT
        ST_AddBand(NULL::raster, '/home/pele/devel/geo/postgis-git/raster/test/regress/ ↩
            loader/Projected.tif', NULL::int[]) AS rast
)
SELECT
    1 AS query,
    *
FROM ST_BandMetadata(
    (SELECT rast FROM foo),
    ARRAY[1,3,2]::int[]
)
UNION ALL
SELECT
    2,
```

```
    *
FROM ST_BandMetadata(
    (
        SELECT
            ST_SetBandIndex(
                rast,
                2,
                1
            ) AS rast
        FROM foo
    ),
    ARRAY[1,3,2]::int[]
)
ORDER BY 1, 2;

 query | bandnum | pixeltype | nodatavalue | isoutdb |  ↩
                                        path                                        |  ↩
     outdbbandnum
-------+---------+-----------+-------------+---------+-----------------------------------------------

     1 |       1 | 8BUI      |             | t       | /home/pele/devel/geo/postgis-git/ ↩
         raster/test/regress/loader/Projected.tif       |            1
     1 |       2 | 8BUI      |             | t       | /home/pele/devel/geo/postgis-git/ ↩
         raster/test/regress/loader/Projected.tif       |            2
     1 |       3 | 8BUI      |             | t       | /home/pele/devel/geo/postgis-git/ ↩
         raster/test/regress/loader/Projected.tif       |            3
     2 |       1 | 8BUI      |             | t       | /home/pele/devel/geo/postgis-git/ ↩
         raster/test/regress/loader/Projected.tif       |            1
     2 |       2 | 8BUI      |             | t       | /home/pele/devel/geo/postgis-git/ ↩
    raster/test/regress/loader/Projected.tif       |            1
     2 |       3 | 8BUI      |             | t       | /home/pele/devel/geo/postgis-git/ ↩
         raster/test/regress/loader/Projected.tif       |            3
```

Veja também

ST_BandMetaData, ST_SetBandPath

10.9 Análises e Estatísticas de Banda Raster

10.9.1 ST_Count

ST_Count — Retorna o número de pixeis em uma banda dada de um raster ou cobertura raster. Se nenhuma banda for especificada, o padrão é usar a banda 1. Se exclude_nodata_value for verdade, contará somente pixeis que não são iguais ao valor nodata.

Synopsis

bigint **ST_Count**(raster rast, integer nband=1, boolean exclude_nodata_value=true);
bigint **ST_Count**(raster rast, boolean exclude_nodata_value);

Descrição

Retorna o número de pixeis em uma banda de um raster ou cobertura raster. Se nenhuma banda foi especificada `nband` usa-se 1.

Note
Se `exclude_nodata_value` for verdade, contará apenas pixeis com valor diferente do valor `nodata` do raster. `exclude_nodata_value` é falso para contar todos os pixeis.

As variantes ST_Count(rastertable, rastercolumn, ...) são deprecadas como da 2.2.0. Ao contrário, use: ST_CountAgg.

Disponibilidade: 2.0.0

Exemplos

```
--example will count all pixels not 249 and one will count all pixels.  --
SELECT rid, ST_Count(ST_SetBandNoDataValue(rast,249)) As exclude_nodata,
        ST_Count(ST_SetBandNoDataValue(rast,249),false) As include_nodata
    FROM dummy_rast WHERE rid=2;

rid | exclude_nodata | include_nodata
-----+----------------+----------------
   2 |             23 |             25
```

Veja também

ST_CountAgg, ST_SummaryStats, ST_SetBandNoDataValue

10.9.2 ST_CountAgg

ST_CountAgg — Agregado. Retorna o número de pixeis em uma banda dada de um raster ou cobertura raster. Se nenhuma banda for especificada, o padrão é usar a banda 1. Se exclude_nodata_value for verdade, contará somente pixeis que são diferentes ao valor NODATA.

Synopsis

bigint **ST_CountAgg**(raster rast, integer nband, boolean exclude_nodata_value, double precision sample_percent);
bigint **ST_CountAgg**(raster rast, integer nband, boolean exclude_nodata_value);
bigint **ST_CountAgg**(raster rast, boolean exclude_nodata_value);

Descrição

Retorna o número de pixeis em uma banda de um conjunto de rasters. Se nenhuma banda foi especificada `nband` usa-se 1.

Se `exclude_nodata_value` for verdade, contará apenas pixeis com valor diferente do valor `NODATA` do raster. `exclude_nodata` é falso para contar todos os pixeis.

Por padrão irá tomar todos os pixeis. Para obter uma resposta mais rápida, coloque `sample_percent` no valor entre zero (0) e um (1)

Disponibilidade: 2.2.0

Exemplos

```
WITH foo AS (
    SELECT
        rast.rast
    FROM (
        SELECT ST_SetValue(
            ST_SetValue(
                ST_SetValue(
                    ST_AddBand(
                        ST_MakeEmptyRaster(10, 10, 10, 10, 2, 2, 0, 0,0)
                        , 1, '64BF', 0, 0
                    )
                    , 1, 1, 1, -10
                )
                , 1, 5, 4, 0
            )
            , 1, 5, 5, 3.14159
        ) AS rast
    ) AS rast
    FULL JOIN (
        SELECT generate_series(1, 10) AS id
    ) AS id
        ON 1 = 1
)
SELECT
    ST_CountAgg(rast, 1, TRUE)
FROM foo;

 st_countagg
-------------
          20
(1 row)
```

Veja também

ST_Count, ST_SummaryStats, ST_SetBandNoDataValue

10.9.3 ST_Histogram

ST_Histogram — Retorna um conjunto de registros que resumem um raster ou distribuição de dados de cobertura raster intervalos bin separados. O número de bins é auto calculado.

Synopsis

SETOF record **ST_Histogram**(raster rast, integer nband=1, boolean exclude_nodata_value=true, integer bins=autocomputed, double precision[] width=NULL, boolean right=false);
SETOF record **ST_Histogram**(raster rast, integer nband, integer bins, double precision[] width=NULL, boolean right=false);
SETOF record **ST_Histogram**(raster rast, integer nband, boolean exclude_nodata_value, integer bins, boolean right);
SETOF record **ST_Histogram**(raster rast, integer nband, integer bins, boolean right);

Descrição

retorna um conjunto de registros de porcentagens min, max, count, para uma banda raster dada para cada bin. Se nenhuma banda for especificada `nband` usa-se 1.

Note
Por padrão só considera valores de pixeis diferentes do valor `nodata`. `exclude_nodata_value` é falso para contar todos os pixeis.

width **double precision[]** largura: um arranjo indicando a largura de cada categoria/bin. Se o número de bins for maior que o número de larguras, elas são repetidas.

Exemplo: 9 bins, larguras são [a, b, c] terão a saída como [a, b, c, a, b, c, a, b, c]

bins **integer** Número de fugas -- este é o número de registros que terá de volta da função especificada. Se não especificado, o número de fugas é auto calculado.

right **boolean** calcula o histograma da direita ao invés do da esquerda (padrão). Isto altera o critério de avaliar um valor x de [a, b) para (a, b]

Changed: 3.1.0 Removed ST_Histogram(table_name, column_name) variant.

Disponibilidade: 2.0.0

Exemplo: Única raster tile - calcula histogramas para bandas 1, 2, 3 e auto calcula bins.

```
SELECT band, (stats).*
FROM (SELECT rid, band, ST_Histogram(rast, band) As stats
    FROM dummy_rast CROSS JOIN generate_series(1,3) As band
     WHERE rid=2) As foo;

 band |  min  |  max  | count | percent
------+-------+-------+-------+---------
    1 |   249 |   250 |     2 |    0.08
    1 |   250 |   251 |     2 |    0.08
    1 |   251 |   252 |     1 |    0.04
    1 |   252 |   253 |     2 |    0.08
    1 |   253 |   254 |    18 |    0.72
    2 |    78 | 113.2 |    11 |    0.44
    2 | 113.2 | 148.4 |     4 |    0.16
    2 | 148.4 | 183.6 |     4 |    0.16
    2 | 183.6 | 218.8 |     1 |    0.04
    2 | 218.8 |   254 |     5 |     0.2
    3 |    62 | 100.4 |    11 |    0.44
    3 | 100.4 | 138.8 |     5 |     0.2
    3 | 138.8 | 177.2 |     4 |    0.16
    3 | 177.2 | 215.6 |     1 |    0.04
    3 | 215.6 |   254 |     4 |    0.16
```

Exemplo: Apenas banda 2 mas para 6 bins

```
SELECT (stats).*
FROM (SELECT rid, ST_Histogram(rast, 2,6) As stats
    FROM dummy_rast
     WHERE rid=2) As foo;

    min     |    max     | count | percent
------------+------------+-------+---------
         78 | 107.333333 |     9 |    0.36
 107.333333 | 136.666667 |     6 |    0.24
 136.666667 |        166 |     0 |       0
        166 | 195.333333 |     4 |    0.16
```

```
 195.333333 | 224.666667 |     1 |    0.04
 224.666667 |        254 |     5 |     0.2
(6 rows)

-- Same as previous but we explicitly control the pixel value range of each bin.
SELECT (stats).*
FROM (SELECT rid, ST_Histogram(rast, 2,6,ARRAY[0.5,1,4,100,5]) As stats
    FROM dummy_rast
     WHERE rid=2) As foo;

  min   |  max   | count | percent
--------+--------+-------+----------
    78  |  78.5  |     1 |     0.08
  78.5  |  79.5  |     1 |     0.04
  79.5  |  83.5  |     0 |        0
  83.5  | 183.5  |    17 |   0.0068
 183.5  | 188.5  |     0 |        0
 188.5  |   254  |     6 | 0.003664
(6 rows)
```

Veja também

ST_Count, ST_SummaryStats, ST_SummaryStatsAgg

10.9.4 ST_Quantile

ST_Quantile — Calcula quantiles para um raster ou cobertura de tabela raster no contexto da amostra ou população. Assim, um valor poderia ser examinado para estar na porcentagem 25%, 50%, 75% do raster.

Synopsis

SETOF record **ST_Quantile**(raster rast, integer nband=1, boolean exclude_nodata_value=true, double precision[] quantiles=NULL);
SETOF record **ST_Quantile**(raster rast, double precision[] quantiles);
SETOF record **ST_Quantile**(raster rast, integer nband, double precision[] quantiles);
double precision **ST_Quantile**(raster rast, double precision quantile);
double precision **ST_Quantile**(raster rast, boolean exclude_nodata_value, double precision quantile=NULL);
double precision **ST_Quantile**(raster rast, integer nband, double precision quantile);
double precision **ST_Quantile**(raster rast, integer nband, boolean exclude_nodata_value, double precision quantile);
double precision **ST_Quantile**(raster rast, integer nband, double precision quantile);

Descrição

Calcula quantiles para um raster ou cobertura de tabela raster no contexto da amostra ou população. Assim, um valor poderia ser examinado para estar na porcentagem 25%, 50%, 75% do raster.

Note
Se `exclude_nodata_value` for falso, contará também pixeis sem dados.

Changed: 3.1.0 Removed ST_Quantile(table_name, column_name) variant.

Disponibilidade: 2.0.0

Exemplos

```
UPDATE dummy_rast SET rast = ST_SetBandNoDataValue(rast,249) WHERE rid=2;
--Example will consider only pixels of band 1 that are not 249 and in named quantiles --

SELECT (pvq).*
FROM (SELECT ST_Quantile(rast, ARRAY[0.25,0.75]) As pvq
    FROM dummy_rast WHERE rid=2) As foo
    ORDER BY (pvq).quantile;

 quantile | value
----------+-------
     0.25 |   253
     0.75 |   254

SELECT ST_Quantile(rast, 0.75) As value
    FROM dummy_rast WHERE rid=2;

value
------
  254
```

```
--real live example.  Quantile of all pixels in band 2 intersecting a geometry
SELECT rid, (ST_Quantile(rast,2)).* As pvc
    FROM o_4_boston
        WHERE ST_Intersects(rast,
            ST_GeomFromText('POLYGON((224486 892151,224486 892200,224706 892200,224706  ↩
                892151,224486 892151))',26986)
            )
ORDER BY value, quantile,rid
;


 rid | quantile | value
-----+----------+-------
   1 |        0 |     0
   2 |        0 |     0
  14 |        0 |     1
  15 |        0 |     2
  14 |     0.25 |    37
   1 |     0.25 |    42
  15 |     0.25 |    47
   2 |     0.25 |    50
  14 |      0.5 |    56
   1 |      0.5 |    64
  15 |      0.5 |    66
   2 |      0.5 |    77
  14 |     0.75 |    81
  15 |     0.75 |    87
   1 |     0.75 |    94
   2 |     0.75 |   106
  14 |        1 |   199
   1 |        1 |   244
   2 |        1 |   255
  15 |        1 |   255
```

Veja também

ST_Count, ST_SummaryStats, ST_SummaryStatsAgg, ST_SetBandNoDataValue

10.9.5 ST_SummaryStats

ST_SummaryStats — Retorna as estatísticas resumidas consistindo de count, sum, mean, stddev, min, max para uma dada banda raster de um raster ou cobertura raster. A banda 1 é assumida se nenhuma banda for especificada.

Synopsis

summarystats **ST_SummaryStats**(raster rast, boolean exclude_nodata_value);
summarystats **ST_SummaryStats**(raster rast, integer nband, boolean exclude_nodata_value);

Descrição

Retorna summarystats consistindo de count, sum, mean, stddev, min, max para uma dada banda raster de um raster ou cobertura raster. Se nenhuma banda for especificada `nband` usa-se a 1.

Note
Por padrão só considera valores de pixeis diferentes do valor `nodata`. `exclude_nodata_value` é falso para contar todos os pixeis.

Note
Por padrão irá tomar todos os pixeis. Para obter uma resposta mais rápida, use `sample_percent` para menor que 1

As variantes ST_SummaryStats(rastertable, rastercolumn, ...) são deprecadas como da 2.2.0. Ao contrário, use: ST_SummaryStatsAgg.

Disponibilidade: 2.0.0

Exemplo: Única tile raster

```
SELECT rid, band, (stats).*
FROM (SELECT rid, band, ST_SummaryStats(rast, band) As stats
    FROM dummy_rast CROSS JOIN generate_series(1,3) As band
     WHERE rid=2) As foo;

 rid | band | count | sum  |    mean    |  stddev   | min | max
-----+------+-------+------+------------+-----------+-----+-----
   2 |    1 |    23 | 5821 | 253.086957 |  1.248061 | 250 | 254
   2 |    2 |    25 | 3682 |     147.28 | 59.862188 |  78 | 254
   2 |    3 |    25 | 3290 |      131.6 | 61.647384 |  62 | 254
```

Exemplo: Resuma pixeis que intersectam construções de interesse

Este exemplo tomou 574ms no PostGIS windows 64-bit com todas as construções de Boston e tiles aéreas (cada uma com 150x150 pixeis ~ 134,000 tiles), ~102,000 registros de construções

```
WITH
-- our features of interest
   feat AS (SELECT gid As building_id, geom_26986 As geom FROM buildings AS b
    WHERE gid IN(100, 103,150)
   ),
-- clip band 2 of raster tiles to boundaries of builds
```

```
-- then get stats for these clipped regions
   b_stats AS
    (SELECT  building_id, (stats).*
FROM (SELECT building_id, ST_SummaryStats(ST_Clip(rast,2,geom)) As stats
    FROM aerials.boston
        INNER JOIN feat
    ON ST_Intersects(feat.geom,rast)
 ) As foo
 )
-- finally summarize stats
SELECT building_id, SUM(count) As num_pixels
  , MIN(min) As min_pval
  ,  MAX(max) As max_pval
  , SUM(mean*count)/SUM(count) As avg_pval
    FROM b_stats
 WHERE count
> 0
    GROUP BY building_id
    ORDER BY building_id;
 building_id | num_pixels | min_pval | max_pval |     avg_pval
-------------+------------+----------+----------+------------------
         100 |       1090 |        1 |      255 | 61.0697247706422
         103 |        655 |        7 |      182 | 70.5038167938931
         150 |        895 |        2 |      252 | 185.642458100559
```

Exemplo: Cobertura raster

```
-- stats for each band --
SELECT band, (stats).*
FROM (SELECT band, ST_SummaryStats('o_4_boston','rast', band) As stats
    FROM generate_series(1,3) As band) As foo;

 band |  count  |  sum   |       mean       |      stddev      | min | max
------+---------+--------+------------------+------------------+-----+-----
    1 | 8450000 | 725799 | 82.7064349112426 | 45.6800222638537 |   0 | 255
    2 | 8450000 | 700487 | 81.4197705325444 | 44.2161184161765 |   0 | 255
    3 | 8450000 | 575943 |  74.682739408284 | 44.2143885481407 |   0 | 255

-- For a table -- will get better speed if set sampling to less than 100%
-- Here we set to 25% and get a much faster answer
SELECT band, (stats).*
FROM (SELECT band, ST_SummaryStats('o_4_boston','rast', band,true,0.25) As stats
    FROM generate_series(1,3) As band) As foo;

 band |  count  |  sum   |       mean       |      stddev      | min | max
------+---------+--------+------------------+------------------+-----+-----
    1 | 2112500 | 180686 | 82.6890480473373 | 45.6961043857248 |   0 | 255
    2 | 2112500 | 174571 |  81.448503668639 | 44.2252623171821 |   0 | 255
    3 | 2112500 | 144364 | 74.6765884023669 | 44.2014869384578 |   0 | 255
```

Veja também

summarystats, ST_SummaryStatsAgg, ST_Count, ST_Clip

10.9.6 ST_SummaryStatsAgg

ST_SummaryStatsAgg — Agregado. Retorna as estatísticas resumidas consistindo de count, sum, mean, stddev, min, max para uma dada banda raster de um conjunto de rasters. A banda 1 é assumida se nenhuma banda for especificada.

Synopsis

summarystats **ST_SummaryStatsAgg**(raster rast, integer nband, boolean exclude_nodata_value, double precision sample_percent);
summarystats **ST_SummaryStatsAgg**(raster rast, boolean exclude_nodata_value, double precision sample_percent);
summarystats **ST_SummaryStatsAgg**(raster rast, integer nband, boolean exclude_nodata_value);

Descrição

Retorna summarystats consistindo de count, sum, mean, stddev, min, max para uma dada banda raster de um raster ou cobertura raster. Se nenhuma banda for especificada `nband` usa-se a 1.

Note
Por padrão só considera valores de pixeis diferentes do valor `NODATA`. `exclude_nodata_value` é falso para contar todos os pixeis.

Note
Por padrão irá tomar todos os pixeis. Para obter uma resposta mais rápida, coloque `sample_percent` no valor entre zero 0 e um 1

Disponibilidade: 2.2.0

Exemplos

```
WITH foo AS (
    SELECT
        rast.rast
    FROM (
        SELECT ST_SetValue(
            ST_SetValue(
                ST_SetValue(
                    ST_AddBand(
                        ST_MakeEmptyRaster(10, 10, 10, 10, 2, 2, 0, 0,0)
                        , 1, '64BF', 0, 0
                    )
                    , 1, 1, 1, -10
                )
                , 1, 5, 4, 0
            )
            , 1, 5, 5, 3.14159
        ) AS rast
    ) AS rast
    FULL JOIN (
        SELECT generate_series(1, 10) AS id
    ) AS id
        ON 1 = 1
)
SELECT
    (stats).count,
    round((stats).sum::numeric, 3),
    round((stats).mean::numeric, 3),
    round((stats).stddev::numeric, 3),
    round((stats).min::numeric, 3),
    round((stats).max::numeric, 3)
FROM (
```

```
    SELECT
        ST_SummaryStatsAgg(rast, 1, TRUE, 1) AS stats
    FROM foo
) bar;

 count |  round   | round  | round |  round   | round
-------+---------+--------+-------+---------+-------
    20 | -68.584 | -3.429 | 6.571 | -10.000 | 3.142
(1 row)
```

Veja também

summarystats, ST_SummaryStats, ST_Count, ST_Clip

10.9.7 ST_ValueCount

ST_ValueCount — Retorna o conjunto de registros contendo uma banda pixel de valor e conta do número de pixeis em uma dada banda de um raster (ou uma cobertura raster) que tem um dado conjunto de valores. Usa-se a banda 1 se nenhuma for especificada. Por padrão pixeis de valor nodata não são contados. Todos os outros valores no pixel são saída e os valores de pixeis são arredondados para o inteiro mais próximo.

Synopsis

SETOF record **ST_ValueCount**(raster rast, integer nband=1, boolean exclude_nodata_value=true, double precision[] searchvalues=NULL, double precision roundto=0, double precision OUT value, integer OUT count);
SETOF record **ST_ValueCount**(raster rast, integer nband, double precision[] searchvalues, double precision roundto=0, double precision OUT value, integer OUT count);
SETOF record **ST_ValueCount**(raster rast, double precision[] searchvalues, double precision roundto=0, double precision OUT value, integer OUT count);
bigint **ST_ValueCount**(raster rast, double precision searchvalue, double precision roundto=0);
bigint **ST_ValueCount**(raster rast, integer nband, boolean exclude_nodata_value, double precision searchvalue, double precision roundto=0);
bigint **ST_ValueCount**(raster rast, integer nband, double precision searchvalue, double precision roundto=0);
SETOF record **ST_ValueCount**(text rastertable, text rastercolumn, integer nband=1, boolean exclude_nodata_value=true, double precision[] searchvalues=NULL, double precision roundto=0, double precision OUT value, integer OUT count);
SETOF record **ST_ValueCount**(text rastertable, text rastercolumn, double precision[] searchvalues, double precision roundto=0, double precision OUT value, integer OUT count);
SETOF record **ST_ValueCount**(text rastertable, text rastercolumn, integer nband, double precision[] searchvalues, double precision roundto=0, double precision OUT value, integer OUT count);
bigint**ST_ValueCount**(text rastertable, text rastercolumn, integer nband, boolean exclude_nodata_value, double precision searchvalue, double precision roundto=0);
bigint **ST_ValueCount**(text rastertable, text rastercolumn, double precision searchvalue, double precision roundto=0);
bigint **ST_ValueCount**(text rastertable, text rastercolumn, integer nband, double precision searchvalue, double precision roundto=0);

Descrição

Retorna um conjunto de registros com colunas `value count` que contêm o valor da banda pixel e soma de pixeis na tile raster ou cobertura raster da banda selecionada.

Se nenhuma banda for especificada `nband` usa-se 1. Se nenhum `searchvalues` for especificado, retornarão pixeis com valores encontrados no raster ou cobertura raster. Se um valor de pesquisa for dado, retornará um inteiro em vez de registros indicando a soma de pixeis que têm aquele valor de banda pixel

Note

Se `exclude_nodata_value` for falso, contará também pixeis sem dados.

Disponibilidade: 2.0.0

Exemplos

```
UPDATE dummy_rast SET rast = ST_SetBandNoDataValue(rast,249) WHERE rid=2;
--Example will count only pixels of band 1 that are not 249. --

SELECT (pvc).*
FROM (SELECT ST_ValueCount(rast) As pvc
    FROM dummy_rast WHERE rid=2) As foo
    ORDER BY (pvc).value;

 value | count
-------+-------
   250 |     2
   251 |     1
   252 |     2
   253 |     6
   254 |    12

-- Example will coount all pixels of band 1 including 249 --
SELECT (pvc).*
FROM (SELECT ST_ValueCount(rast,1,false) As pvc
    FROM dummy_rast WHERE rid=2) As foo
    ORDER BY (pvc).value;

 value | count
-------+-------
   249 |     2
   250 |     2
   251 |     1
   252 |     2
   253 |     6
   254 |    12

-- Example will count only non-nodata value pixels of band 2
SELECT (pvc).*
FROM (SELECT ST_ValueCount(rast,2) As pvc
    FROM dummy_rast WHERE rid=2) As foo
    ORDER BY (pvc).value;
 value | count
-------+-------
    78 |     1
    79 |     1
    88 |     1
    89 |     1
    96 |     1
    97 |     1
    98 |     1
    99 |     2
   112 |     2
:
```

```
--real live example.  Count all the pixels in an aerial raster tile band 2 intersecting a  ↩
    geometry
-- and return only the pixel band values that have a count
> 500
SELECT (pvc).value, SUM((pvc).count) As total
FROM (SELECT ST_ValueCount(rast,2) As pvc
    FROM o_4_boston
        WHERE ST_Intersects(rast,
            ST_GeomFromText('POLYGON((224486 892151,224486 892200,224706 892200,224706  ↩
                892151,224486 892151))',26986)
             )
        ) As foo
    GROUP BY (pvc).value
    HAVING SUM((pvc).count)
> 500
    ORDER BY (pvc).value;

 value | total
-------+-----
    51 | 502
    54 | 521
```

```
-- Just return count of pixels in each raster tile that have value of 100 of tiles that  ↩
    intersect  a specific geometry --
SELECT rid, ST_ValueCount(rast,2,100) As count
    FROM o_4_boston
        WHERE ST_Intersects(rast,
            ST_GeomFromText('POLYGON((224486 892151,224486 892200,224706 892200,224706  ↩
                892151,224486 892151))',26986)
             ) ;

 rid | count
-----+-------
   1 |    56
   2 |    95
  14 |    37
  15 |    64
```

Veja também

ST_Count, ST_SetBandNoDataValue

10.10 Raster Inputs

10.10.1 ST_RastFromWKB

ST_RastFromWKB — Return a raster value from a Well-Known Binary (WKB) raster.

Synopsis

raster **ST_RastFromWKB**(bytea wkb);

Descrição

Given a Well-Known Binary (WKB) raster, return a raster.

Availability: 2.5.0

Exemplos

```
SELECT (ST_Metadata(
    ST_RastFromWKB(
        '\001\000\000\000\000\000\000\000\000\000\000@\000\000\000\000\000\000\010@ ↩
            \000\000\000\000\000\000\340?\000\000\000\000\000\000\340?\000\000\000\000\000\000\000\0
            bytea
    )
)).* AS metadata;

 upperleftx | upperlefty | width | height | scalex | scaley | skewx | skewy | srid |  ↩
    numbands
------------+------------+-------+--------+--------+--------+-------+-------+------+---------- ↩

        0.5 |        0.5 |    10 |     20 |      2 |      3 |     0 |     0 |   10 |  ↩
                  0
```

Veja também

ST_MetaData, ST_RastFromHexWKB, ST_AsBinary/ST_AsWKB, ST_AsHexWKB

10.10.2 ST_RastFromHexWKB

ST_RastFromHexWKB — Return a raster value from a Hex representation of Well-Known Binary (WKB) raster.

Synopsis

raster **ST_RastFromHexWKB**(text wkb);

Descrição

Given a Well-Known Binary (WKB) raster in Hex representation, return a raster.

Availability: 2.5.0

Exemplos

```
SELECT (ST_Metadata(
    ST_RastFromHexWKB(
        '01000000000000000000000040000000000000084000000000000000 ↩
            E03F000000000000E03F0000000000000000000000000000000A0000000A001400'
    )
)).* AS metadata;

 upperleftx | upperlefty | width | height | scalex | scaley | skewx | skewy | srid |  ↩
    numbands
------------+------------+-------+--------+--------+--------+-------+-------+------+---------- ↩

        0.5 |        0.5 |    10 |     20 |      2 |      3 |     0 |     0 |   10 |  ↩
                  0
```

Veja também

ST_MetaData, ST_RastFromWKB, ST_AsBinary/ST_AsWKB, ST_AsHexWKB

10.11 Raster Outputs

10.11.1 ST_AsBinary/ST_AsWKB

ST_AsBinary/ST_AsWKB — Return the Well-Known Binary (WKB) representation of the raster.

Synopsis

bytea **ST_AsBinary**(raster rast, boolean outasin=FALSE);
bytea **ST_AsWKB**(raster rast, boolean outasin=FALSE);

Descrição

Returns the Binary representation of the raster. If `outasin` is TRUE, out-db bands are treated as in-db. Refer to raster/doc/RFC2-WellKnownBinaryFormat located in the PostGIS source folder for details of the representation.

Isso é útil em cursores binários para tirar dados do banco de dados sem converter para uma representação de string.

Note
Por padrão, saída WKB contém o caminho do arquivo externo para bandas fora do banco de dados. Se o cliente não tem acesso ao arquivo raster secreto uma banda fora do banco de dados, `outasin` é VERDADE.

Melhorias: 2.1.0 Adição da `outasin`

Enhanced: 2.5.0 Addition of `ST_AsWKB`

Exemplos

```
SELECT ST_AsBinary(rast) As rastbin FROM dummy_rast WHERE rid=1;

                     rastbin
---------------------------------------------------------------------------
\001\000\000\000\000\000\000\000\000\000\000\000@\000\000\000\000\000\000\010@ ↩
    \000\000\000\000\000\000\340?\000\000\000\000\000\000\340?\000\000\000\000\000\000\000\000\00
```

Veja também

ST_RastFromWKB, ST_AsHexWKB

10.11.2 ST_AsHexWKB

ST_AsHexWKB — Return the Well-Known Binary (WKB) in Hex representation of the raster.

Synopsis

bytea **ST_AsHexWKB**(raster rast, boolean outasin=FALSE);

Descrição

Returns the Binary representation in Hex representation of the raster. If `outasin` is TRUE, out-db bands are treated as in-db. Refer to raster/doc/RFC2-WellKnownBinaryFormat located in the PostGIS source folder for details of the representation.

Note
By default, Hex WKB output contains the external file path for out-db bands. If the client does not have access to the raster file underlying an out-db band, set `outasin` to TRUE.

Availability: 2.5.0

Exemplos

```
SELECT ST_AsHexWKB(rast) As rastbin FROM dummy_rast WHERE rid=1;

                                                                st_ashexwkb
------------------------------------------------------------------------------------------------------------------------

 01000000000000000000000040000000000000084000000000000000 ↩
    E03F000000000000E03F000000000000000000000000000000000A0000000A001400
```

Veja também

ST_RastFromHexWKB, ST_AsBinary/ST_AsWKB

10.11.3 ST_AsGDALRaster

ST_AsGDALRaster — Return the raster tile in the designated GDAL Raster format. Raster formats are one of those supported by your compiled library. Use ST_GDALDrivers() to get a list of formats supported by your library.

Synopsis

bytea **ST_AsGDALRaster**(raster rast, text format, text[] options=NULL, integer srid=sameassource);

Descrição

Retorna a tile raster no formato designado. Os argumentos estão listados abaixo:

- `format` formato para saída. Isso depende dos drivers compilados na sua biblioteca libgdal. Estão normalmente disponíveis 'JPEG', 'GTIff', 'PNG'. Use ST_GDALDrivers para conseguir uma lista dos formatos suportados por sua biblioteca.
- `options` texto arranjo de opções GDAL. As opções válidas são dependentes do formato. Recorra a GDAL Raster format options para mais detalhes.
- `srs` O proj4text ou srtext (do spatial_ref_sys) para embutir na imagem

Disponibilidade: 2.0.0 - requer GDAL >= 1.6.0.

JPEG Output Example, multiple tiles as single raster

```
SELECT ST_AsGDALRaster(ST_Union(rast), 'JPEG', ARRAY['QUALITY=50']) As rastjpg
FROM dummy_rast
WHERE rast && ST_MakeEnvelope(10, 10, 11, 11);
```

Using PostgreSQL Large Object Support to export raster

One way to export raster into another format is using PostgreSQL large object export functions. We'lll repeat the prior example but also exporting. Note for this you'll need to have super user access to db since it uses server side lo functions. It will also export to path on server network. If you need export locally, use the psql equivalent lo_ functions which export to the local file system instead of the server file system.

```
DROP TABLE IF EXISTS tmp_out ;

CREATE TABLE tmp_out AS
SELECT lo_from_bytea(0,
       ST_AsGDALRaster(ST_Union(rast), 'JPEG', ARRAY['QUALITY=50'])
        ) AS loid
  FROM dummy_rast
WHERE rast && ST_MakeEnvelope(10, 10, 11, 11);

SELECT lo_export(loid, '/tmp/dummy.jpg')
   FROM tmp_out;

SELECT lo_unlink(loid)
  FROM tmp_out;
```

GTIFF Output Examples

```
SELECT ST_AsGDALRaster(rast, 'GTiff') As rastjpg
FROM dummy_rast WHERE rid=2;

-- Out GeoTiff with jpeg compression, 90% quality
SELECT ST_AsGDALRaster(rast, 'GTiff',
  ARRAY['COMPRESS=JPEG', 'JPEG_QUALITY=90'],
  4269) As rasttiff
FROM dummy_rast WHERE rid=2;
```

Veja também

Section 9.3, ST_GDALDrivers, ST_SRID

10.11.4 ST_AsJPEG

ST_AsJPEG — Retorna as bandas tile raster selecionadas como uma única Joint Photographic Exports Group (JPEG) image (byte arranjo). Se nenhuma banda for especificada e 1 ou mais que 3 bandas, então somente a primeira banda é usada. Se somente 3 bandas, então todas as 3 bandas serão usadas para mapear par RGB.

Synopsis

bytea **ST_AsJPEG**(raster rast, text[] options=NULL);
bytea **ST_AsJPEG**(raster rast, integer nband, integer quality);
bytea **ST_AsJPEG**(raster rast, integer nband, text[] options=NULL);
bytea **ST_AsJPEG**(raster rast, integer[] nbands, text[] options=NULL);
bytea **ST_AsJPEG**(raster rast, integer[] nbands, integer quality);

Descrição

Retorna as bandas selecionadas do raster como uma única Joint Photographic Exports Group Image (JPEG). Use ST_AsGDALRaster se precisar exportar como tipos raster menos comuns. Se nenhuma banda for especificada e 1 ou mais que 3 bandas, então somente a primeira é usada. Se 3 bandas, então 3 bandas são usadas. Existem muitas variantes da função com várias opções. Elas estão listadas abaixo:

- `nband` é para exportação de uma única banda.
- `nbands` é um arranjo para exportar (note que o máximo é 3 para JPEG) e a ordem das bandas é RGB. ex.: ARRAY[3,2,1] significa mapa banda 3 para Vermelho, banda 2 para verde e banda 1 para azul.
- `quality` número de 0 a 100. Quanto maior o número mais translúcida a imagem.
- `options` opções de textos Array of GDAL definidas para JPEG (veja em create_options para JPEG ST_GDALDrivers). Para JPEG válido eles são `PROGRESSIVE` ON or OFF e `QUALITY` a range from 0 to 100 and default to 75. Recorra a GDAL Raster format options para mais dealhes.

Disponibilidade: 2.0.0 - requer GDAL >= 1.6.0.

Exemplos: Saída

```
-- output first 3 bands 75% quality
SELECT ST_AsJPEG(rast) As rastjpg
    FROM dummy_rast WHERE rid=2;

-- output only first band as 90% quality
SELECT ST_AsJPEG(rast,1,90) As rastjpg
    FROM dummy_rast WHERE rid=2;

-- output first 3 bands (but make band 2 Red, band 1 green, and band 3 blue, progressive ↩
    and 90% quality
SELECT ST_AsJPEG(rast,ARRAY[2,1,3],ARRAY['QUALITY=90','PROGRESSIVE=ON']) As rastjpg
    FROM dummy_rast WHERE rid=2;
```

Veja também

Section 9.3, ST_GDALDrivers, ST_AsGDALRaster, ST_AsPNG, ST_AsTIFF

10.11.5 ST_AsPNG

ST_AsPNG — Retorna as bandas tile raster selecionadas como um gráfico de rede portátil (PNG) imagem (byte array). Se as bandas raster 1, 3 ou 4 e nenhum banda for especificado, então todas as bandas são usadas. Se mais 2 ou mais que 4 bandas e nenhuma banda forem especificadas, então somente a banda 1 é usada. As bandas são mapeadas para espeço RGB ou RGBA.

Synopsis

bytea **ST_AsPNG**(raster rast, text[] options=NULL);
bytea **ST_AsPNG**(raster rast, integer nband, integer compression);
bytea **ST_AsPNG**(raster rast, integer nband, text[] options=NULL);
bytea **ST_AsPNG**(raster rast, integer[] nbands, integer compression);
bytea **ST_AsPNG**(raster rast, integer[] nbands, text[] options=NULL);

Descrição

Retorna as bandas selecionadas do raster como uma única Portable Network Graphics Image (PNG). Use ST_AsGDALRaster se precisar exportar como os tipo de raster menos comuns. Se nenhuma banda for especificada, então as 3 primeiras bandas serão exportadas. Existem muitas variantes da função com várias opções. Se nenhum `srid`for especificado, o srid do raster é usado. Eles estão listados abaixo:

- `nband` é para exportação de uma única banda.
- `nbands` é um arranjo para exportar (note que o máximo é 4 para JPEG) e a ordem das bandas é RGB. ex.: ARRAY[3,2,1] significa mapa banda 3 para Vermelho, banda 2 para verde e banda 1 para azul.
- `compression` número de 1 a 9. Quanto maior o número melhor a compressão.
- `options` opções de textos do Arranjo do GDAL como definidas para PNG (veja em create_options para PNG da ST_GDALDrivers). Para PNG válido é somente ZLEVEL (porção de tempo para gastar na comprensão -- padrão 6) ex.: ARRAY['ZLEVEL=9']. WORLDFILE não é permitido já que a função teria que gerar duas saídas. Recorra a GDAL Raster format options para mais detalhes.

Disponibilidade: 2.0.0 - requer GDAL >= 1.6.0.

Exemplos

```
SELECT ST_AsPNG(rast) As rastpng
FROM dummy_rast WHERE rid=2;

-- export the first 3 bands and map band 3 to Red, band 1 to Green, band 2 to blue
SELECT ST_AsPNG(rast, ARRAY[3,1,2]) As rastpng
FROM dummy_rast WHERE rid=2;
```

Veja também

ST_AsGDALRaster, ST_ColorMap, ST_GDALDrivers, Section 9.3

10.11.6 ST_AsTIFF

ST_AsTIFF — Return the raster selected bands as a single TIFF image (byte array). If no band is specified or any of specified bands does not exist in the raster, then will try to use all bands.

Synopsis

bytea **ST_AsTIFF**(raster rast, text[] options='', integer srid=sameassource);
bytea **ST_AsTIFF**(raster rast, text compression='', integer srid=sameassource);
bytea **ST_AsTIFF**(raster rast, integer[] nbands, text compression='', integer srid=sameassource);
bytea **ST_AsTIFF**(raster rast, integer[] nbands, text[] options, integer srid=sameassource);

Descrição

Retorna as bandas selecionadas do raster como um formato Tagged Image File Format (TIFF) único. Se nenhuma banda estiver especificada, tentaremos usar todas as bandas. Isto é uma envoltório em torno da ST_AsGDALRaster. Use ST_AsGDALRaster se precisar exportar como tipos raster menos comuns. Existem muitas variantes da função com diversas opções. Se nenhuma texto de referência espacial SRS estiver presente, a referência espacial do raster é usada. Elas estão listadas abaixo:

- `nbands` é um arranjo de bandas para exportar (note que o máximo é 3 para PNG) e a ordem das bandas é RGB. ex.: ARRAY[3,2,1] significa mapear banda 3 para vermelho, banda 2 para verde e banda 1 para azul
- `compression` Expressão de compressão -- JPEG90 (ou algum outro percentual), LZW, JPEG, DEFLATE9.
- `options` text Array of GDAL create options as defined for GTiff (look at create_options for GTiff of ST_GDALDrivers). or refer to GDAL Raster format options for more details.
- `srid` srid do spatial_ref_sys do raster. É usado para popular a informação georreferência

Disponibilidade: 2.0.0 - requer GDAL >= 1.6.0.

Exemplo: Use jpeg compressão 90%

```
SELECT ST_AsTIFF(rast, 'JPEG90') As rasttiff
FROM dummy_rast WHERE rid=2;
```

Veja também

ST_GDALDrivers, ST_AsGDALRaster, ST_SRID

10.12 Processamento Raster

10.12.1 ST_Clip

ST_Clip — Retorna o raster suprimido pela geometria de entrada. Se o número de banda não for especificado, todas as bandas são processadas. Se `crop` não for especificado ou for VERDADE, o raster de saída é cortado.

Synopsis

raster **ST_Clip**(raster rast, integer[] nband, geometry geom, double precision[] nodataval=NULL, boolean crop=TRUE);
raster **ST_Clip**(raster rast, integer nband, geometry geom, double precision nodataval, boolean crop=TRUE);
raster **ST_Clip**(raster rast, integer nband, geometry geom, boolean crop);
raster **ST_Clip**(raster rast, geometry geom, double precision[] nodataval=NULL, boolean crop=TRUE);
raster **ST_Clip**(raster rast, geometry geom, double precision nodataval, boolean crop=TRUE);
raster **ST_Clip**(raster rast, geometry geom, boolean crop);

Descrição

Retorna um raster que é suprimido pela geometria de entrada `geom`. Se o índice de banda não for especificado, todas as bandas são processadas.

Os rasters resultantes da ST_Clip devem ter o valor nodata designado para as áreas suprimidas, um para cada banda. Se nenhum for promovido e o raster de entrada não tiver nenhum valor nodata definido, os valore nodata do raster resultante são ST_MinPossibleValue(ST_BandPixelType(rast, band)). Quando o número de valor nodata no arranjo é menor qua o número de banda, o último no arranjo é usado para as bandas que sobraram. Se o número de valor nodata for maior que o número de banda, os valores extras serão ignorados. Todas as variantes que aceitam um arranjo de valores nodata também aceitam um valor único, que pode ser designado para cada banda.

Se `crop` não for especificado, é verdade, significando que o rasetr de saída é cortado para a intersecção das extensões `geom`e `rast`. Se `crop` for falso, o novo raster tem a mesma extensão que `rast`.

Disponibilidade: 2.0.0

Melhorias: 2.1.0 Reescrito em C

Os exemplos aqui utilizam os dados areais de Massachusetts disponíveis no site MassGIS MassGIS Aerial Orthos. As coordenadas estão no Massachusetts State Plane Meters.

Exemplos: 1 banda suprimindo

```
-- Clip the first band of an aerial tile by a 20 meter buffer.
SELECT ST_Clip(rast, 1,
        ST_Buffer(ST_Centroid(ST_Envelope(rast)),20)
    ) from aerials.boston
WHERE rid = 4;
```

```
-- Demonstrate effect of crop on final dimensions of raster
-- Note how final extent is clipped to that of the geometry
-- if crop = true
SELECT ST_XMax(ST_Envelope(ST_Clip(rast, 1, clipper, true))) As xmax_w_trim,
    ST_XMax(clipper) As xmax_clipper,
    ST_XMax(ST_Envelope(ST_Clip(rast, 1, clipper, false))) As xmax_wo_trim,
    ST_XMax(ST_Envelope(rast)) As xmax_rast_orig
FROM (SELECT rast, ST_Buffer(ST_Centroid(ST_Envelope(rast)),6) As clipper
    FROM aerials.boston
WHERE rid = 6) As foo;

   xmax_w_trim    |   xmax_clipper   |   xmax_wo_trim   |  xmax_rast_orig
------------------+------------------+------------------+------------------
 230657.436173996 | 230657.436173996 | 230666.436173996 | 230666.436173996
```

Tile raster completa antes se suprimir

Depois de suprimir

Exemplos: 1 banda suprimindo sem cortes e adiciona de volta outras bandas inalteradas

```
-- Same example as before, but we need to set crop to false to be able to use ST_AddBand
-- because ST_AddBand requires all bands be the same Width and height
SELECT ST_AddBand(ST_Clip(rast, 1,
        ST_Buffer(ST_Centroid(ST_Envelope(rast)),20),false
    ), ARRAY[ST_Band(rast,2),ST_Band(rast,3)] ) from aerials.boston
WHERE rid = 6;
```

Tile raster completa antes se suprimir

Depois de suprimir - surreal

Exemplos: Suprime todas as bandas

```
-- Clip all bands of an aerial tile by a 20 meter buffer.
-- Only difference is we don't specify a specific band to clip
-- so all bands are clipped
SELECT ST_Clip(rast,
      ST_Buffer(ST_Centroid(ST_Envelope(rast)), 20),
      false
    ) from aerials.boston
WHERE rid = 4;
```

Tile raster completa antes se suprimir

Depois de suprimir

Veja também

ST_AddBand, Funções retorno de mapa algébrico embutido, ST_Intersection

10.12.2 ST_ColorMap

ST_ColorMap — Cria um novo raster de até quatro bandas 8BUI (grayscale, RGB, RGBA) do raster fonte e uma banda específica. A banda 1 usada se não especificado.

Synopsis

raster **ST_ColorMap**(raster rast, integer nband=1, text colormap=grayscale, text method=INTERPOLATE);

raster **ST_ColorMap**(raster rast, text colormap, text method=INTERPOLATE);

Descrição

Aplica um `colormap` à banda na `nband` do `rast` resultando em um novo raster englobado com até quatro bandas 8BUI. O número de bandas 8BUI no novo raster é determinado pelo número de cores componentes definidas no `colormap`.

Se `nband` não for especificado, a banda 1 é assumida.

`colormap` pode ser uma palavra-chave de um colormap pré definido ou um conjunto de linhas definindo o valor e a cor dos componentes.

Palavra-chave válida do `colormap` pré definida:

- `grayscale` ou `greyscale` para uma banda raster 8BUI de tons de cinza.
- `pseudocolor` para quatro bandas raster 8BUI (RGBA) com cores indo de azul para verde e para vermelho.
- `fire` para quatro bandas raster 8BUI (RGBA) com cores indo de preto para vermelho para amarelo claro.
- `bluered` para quatro bandas raster 8BUI (RGBA) com cores indo de azul para branco para vermelho.

Os usuários podem passar um conjunto de entradas (uma por linha) para `colormap` para especificar colormaps personalizados. Cada entrada consiste de cinco valores: o valor de pixel e componentes Vermelho, Verde, Azul, Alfa correspondentes (entre 0 e 255). Valores de porcentagem podem ser usados em vez de valores de pixel onde 0% e 100% são os mínimos e os máximos encontrados na banda raster. Os valores podem ser separados por vírgulas (","), tabs, dois pontos (":") e/ou espaços. O valor do pixel pode ser *nv*, *null* ou *nodata* para o valor NODATA. Um exemplo é fornecido abaixo.

```
5 0 0 0 255
4 100:50 55 255
1 150,100 150 255
0% 255 255 255 255
nv 0 0 0 0
```

A sintaxe do `colormap` é parecida com com a do modo do auxílio de cor do GDAL gdaldem.

Palavras-chave válidas para `method`:

- `INTERPOLATE` para usar interpolação linear para misturar suavemente as cores entre os valores do pixel
- `EXACT` para combinar estritamente somente aqueles valores de pixel encontrados no colormap. Os pixeis cujos valores não combinarem com uma entrada do colormap serão 0 0 0 0 (RGBA)
- `NEAREST` para usar a entrada do colormap cujos valores são mais próximos ao valor do pixel

Note
Uma ótima referência para o colormap é ColorBrewer.

Warning
As bandas resultantes do novo raster não terá nenhum valor NODATA. Use ST_SetBandNoDataValue se precisar de um valor NODATA.

Disponibilidade: 2.1.0

Exemplos

Esta não é uma boa tabela para desfrutar

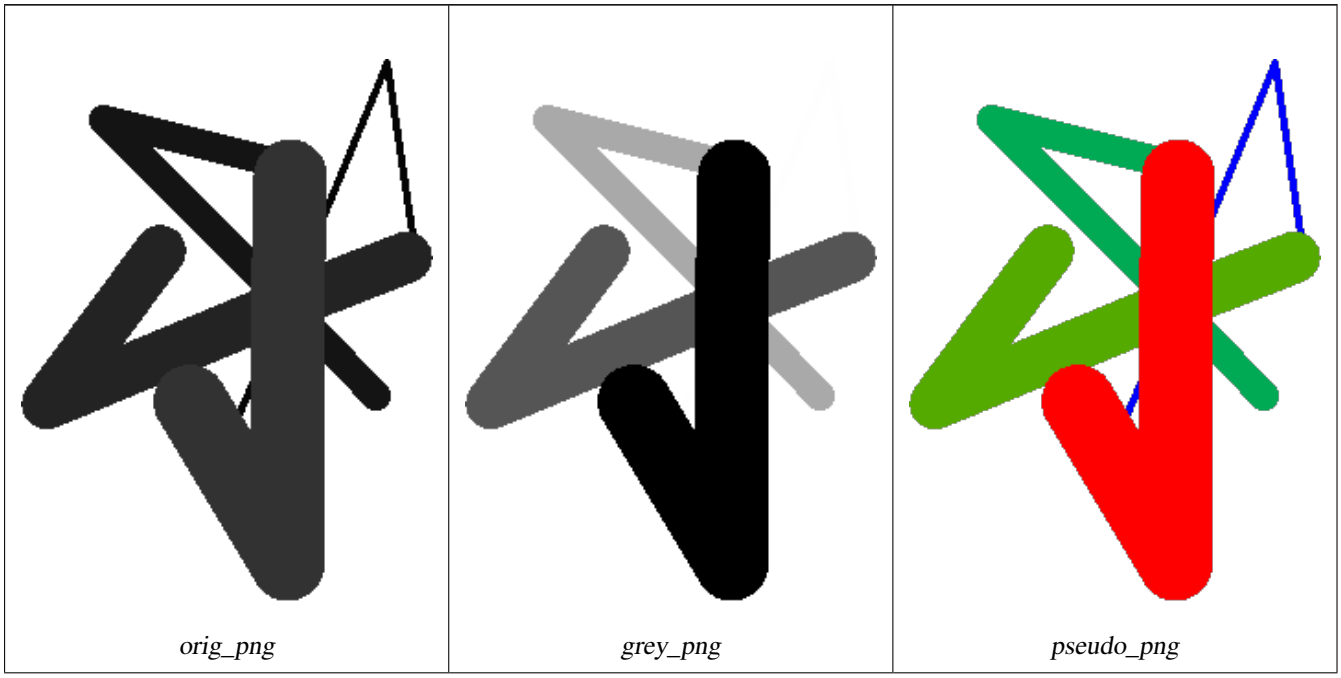
```
-- setup test raster table --
DROP TABLE IF EXISTS funky_shapes;
CREATE TABLE funky_shapes(rast raster);

INSERT INTO funky_shapes(rast)
WITH ref AS (
    SELECT ST_MakeEmptyRaster( 200, 200, 0, 200, 1, -1, 0, 0) AS rast
)
SELECT
    ST_Union(rast)
FROM (
    SELECT
        ST_AsRaster(
            ST_Rotate(
                ST_Buffer(
                    ST_GeomFromText('LINESTRING(0 2,50 50,150 150,125 50)'),
                    i*2
                ),
                pi() * i * 0.125, ST_Point(50,50)
            ),
            ref.rast, '8BUI'::text, i * 5
        ) AS rast
    FROM ref
    CROSS JOIN generate_series(1, 10, 3) AS i
) AS shapes;
```

```
SELECT
    ST_NumBands(rast) As n_orig,
    ST_NumBands(ST_ColorMap(rast,1, 'greyscale')) As ngrey,
    ST_NumBands(ST_ColorMap(rast,1, 'pseudocolor')) As npseudo,
    ST_NumBands(ST_ColorMap(rast,1, 'fire')) As nfire,
    ST_NumBands(ST_ColorMap(rast,1, 'bluered')) As nbluered,
    ST_NumBands(ST_ColorMap(rast,1, '
100% 255   0   0
 80% 160   0   0
 50% 130   0   0
 30%  30   0   0
 20%  60   0   0
  0%   0   0   0
  nv 255 255 255
    ')) As nred
FROM funky_shapes;
```

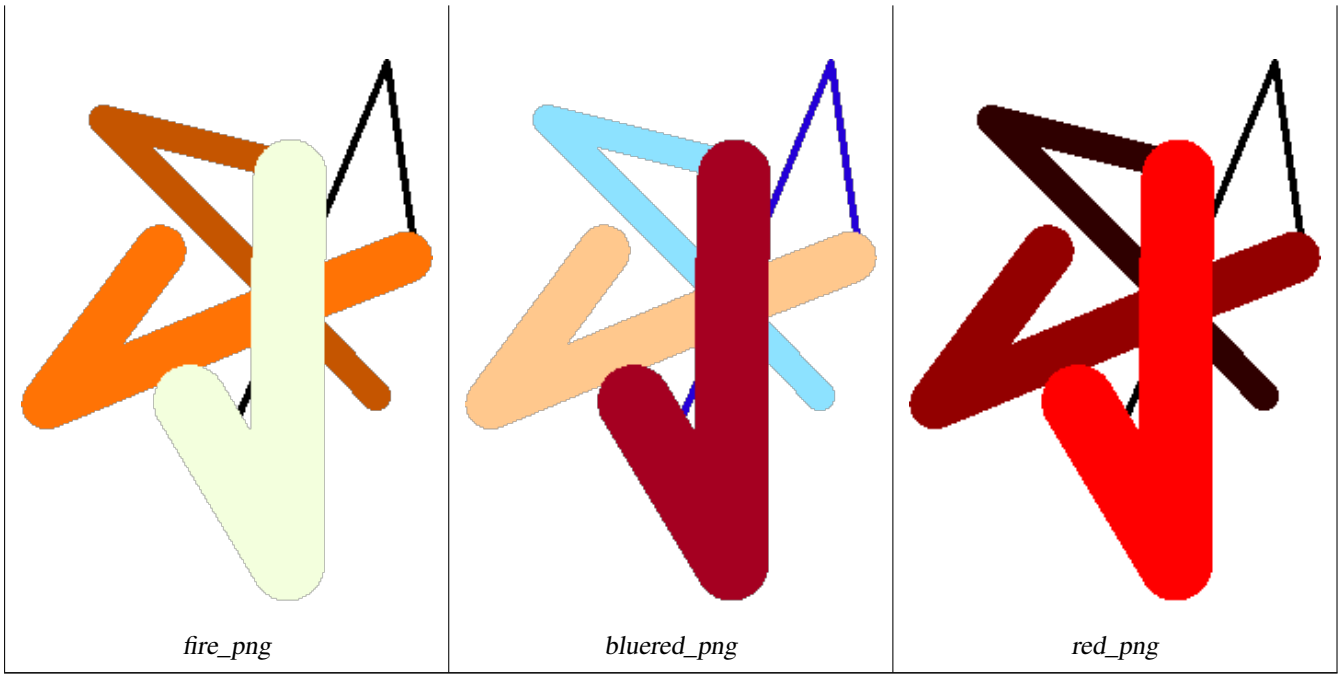
```
n_orig | ngrey | npseudo | nfire | nbluered | nred
--------+-------+---------+-------+----------+------
      1 |     1 |       4 |     4 |        4 |    3
```

Exemplos: Compara cores diferentes no mapa usando ST_AsPNG

```
SELECT
    ST_AsPNG(rast) As orig_png,
    ST_AsPNG(ST_ColorMap(rast,1,'greyscale')) As grey_png,
    ST_AsPNG(ST_ColorMap(rast,1, 'pseudocolor')) As pseudo_png,
    ST_AsPNG(ST_ColorMap(rast,1, 'nfire')) As fire_png,
    ST_AsPNG(ST_ColorMap(rast,1, 'bluered')) As bluered_png,
    ST_AsPNG(ST_ColorMap(rast,1, '
100% 255   0   0
 80% 160   0   0
 50% 130   0   0
 30%  30   0   0
 20%  60   0   0
  0%   0   0   0
  nv 255 255 255
    ')) As red_png
FROM funky_shapes;
```



orig_png | *grey_png* | *pseudo_png*



fire_png | *bluered_png* | *red_png*

Veja também

ST_AsPNG, ST_AsRaster Funções retorno de mapa algébrico embutido, ST_Grayscale ST_NumBands, ST_Reclass, ST_SetBandNoDa ST_Union

10.12.3 ST_Grayscale

ST_Grayscale — Creates a new one-8BUI band raster from the source raster and specified bands representing Red, Green and Blue

Synopsis

(1) raster **ST_Grayscale**(raster rast, integer redband=1, integer greenband=2, integer blueband=3, text extenttype=INTERSECTION);

(2) raster **ST_Grayscale**(rastbandarg[] rastbandargset, text extenttype=INTERSECTION);

Descrição

Create a raster with one 8BUI band given three input bands (from one or more rasters). Any input band whose pixel type is not 8BUI will be reclassified using ST_Reclass.

Note
This function is not like ST_ColorMap with the `grayscale` keyword as ST_ColorMap operates on only one band while this function expects three bands for RGB. This function applies the following equation for converting RGB to Grayscale: 0.2989 * RED + 0.5870 * GREEN + 0.1140 * BLUE

Availability: 2.5.0

Exemplos: Variante 1

```
SET postgis.gdal_enabled_drivers = 'ENABLE_ALL';
SET postgis.enable_outdb_rasters = True;

WITH apple AS (
    SELECT ST_AddBand(
        ST_MakeEmptyRaster(350, 246, 0, 0, 1, -1, 0, 0, 0),
        '/tmp/apple.png'::text,
        NULL::int[]
    ) AS rast
)
SELECT
    ST_AsPNG(rast) AS original_png,
    ST_AsPNG(ST_Grayscale(rast)) AS grayscale_png
FROM apple;
```

original_png

grayscale_png

Exemplos: Variant 2

```
SET postgis.gdal_enabled_drivers = 'ENABLE_ALL';
SET postgis.enable_outdb_rasters = True;

WITH apple AS (
    SELECT ST_AddBand(
        ST_MakeEmptyRaster(350, 246, 0, 0, 1, -1, 0, 0, 0),
        '/tmp/apple.png'::text,
        NULL::int[]
    ) AS rast
)
SELECT
    ST_AsPNG(rast) AS original_png,
    ST_AsPNG(ST_Grayscale(
        ARRAY[
            ROW(rast, 1)::rastbandarg, -- red
            ROW(rast, 2)::rastbandarg, -- green
            ROW(rast, 3)::rastbandarg, -- blue
        ]::rastbandarg[]
    )) AS grayscale_png
FROM apple;
```

Veja também

ST_AsPNG, ST_Reclass, ST_ColorMap

10.12.4 ST_Intersection

ST_Intersection — Retorna uma raster ou conjunto de pares de valores de pixeis de geometria representando a porção dividida de dois rasters ou a interseção geométrica de uma vetorização do raster e uma geometria.

Synopsis

setof geomval **ST_Intersection**(geometry geom, raster rast, integer band_num=1);
setof geomval **ST_Intersection**(raster rast, geometry geom);
setof geomval **ST_Intersection**(raster rast, integer band, geometry geomin);
raster **ST_Intersection**(raster rast1, raster rast2, double precision[] nodataval);
raster **ST_Intersection**(raster rast1, raster rast2, text returnband, double precision[] nodataval);
raster **ST_Intersection**(raster rast1, integer band1, raster rast2, integer band2, double precision[] nodataval);
raster **ST_Intersection**(raster rast1, integer band1, raster rast2, integer band2, text returnband, double precision[] nodataval);

Descrição

Retorna uma raster ou conjunto de pares de valores de pixeis de geometria representando a porção dividida de dois rasters ou a interseção geométrica de uma vetorização do raster e uma geometria.

As primeiras três variantes, retornando um conjunto de geomval, funciona no espaço vetor. Primeiramente, o raster é vetorizado (usando a ST_DumpAsPolygon) dentro de linhas geomval e elas intersectam com a geometria usando a função PostGIS ST_Intersection(geometria, geometria). Somente as geometrias intersectando com uma área de valor nodata de um raster, retornam uma geometria vazia. Normalmente, elas são excluídas dos resultados pelo próprio uso da ST_Intersect na cláusula ONDE.

Você pode acessar a geometria e as partes do valor do conjunto geomval resultante colocando parênteses e adicionando '.geom' ou '.val' no fim da expressão. ex.: (ST_Intersection(rast, geom)).geom

As outras variantes, retornando um raster, funcionam no espaço raster. Elas estão usando a versão de dois raster da ST_MapAlgebraExp para representar a interseção.

A extensão do raster resultante corresponde à interseção geométrica das duas extensões raster. O raster resultante inclui 'BANDA1', 'BANDA2' ou 'AMBAS' as bandas, a seguir o que é passado como o parâmetro `returnband`. As áreas do valor nodata presentes em qualquer banda resultam áreas de valor nodata em todas as bandas do resultado. Em outras palavras, qualquer pixel intersectando com um pixel de valor nodata se torna um pixel de valor nodata no resultado.

Os rasters resultantes da ST_Intersection devem ter um valor nodata designado para áreas que não intersectam. Você pode definir ou substituir o valor nodata para qualquer banda resultante fornecendo um arranjo `nodataval[]` de um ou dois valores nodata, dependendo se solicitou 'BANDA1', 'BANDA2' ou 'AMBAS' as bandas. O primeiro valor no arranjo substitui o valor nodata na primeira banda e o segundo substitui o valor nodata na segunda banda. Se uma banda de entrada não possuir o valor nodata definido e nenhum for fornecido como arranjo, um é escolhido usando a função ST_MinPossibleValue. Todas as variantes que aceitam um arranjo com valor nodata também aceita um único valor que pode ser designado para cada banda pedida.

Em todas as variantes, se nenhum número de banda for especificado, a banda 1 é assumida. Se precisar de uma interseção entre um raster e uma geometria que retorna um raster, recorra a ST_Clip.

Note
Para ter mais controle na extensão resultante ou no que retorna quando encontra um valor nodata, use a versão de dois raster da ST_MapAlgebraExpr.

Note
Para calcular a interseção de uma banda raster com uma geometria em um espaço raster, use ST_Clip. ST_Clip funciona em várias bandas rasters e não retorna uma banda correspondente para uma geometria rasterizada.

Note
A ST_Intersection deveria ser usada em conjunto com a ST_Intersects e um índice na coluna raster e/ou na coluna geométrica.

Melhorias: 2.0.0 - Interseção no espaço raster foi introduzida. Nas versões anteriores pre-2.0.0, somente a interseção apresentada no espaço do vetor era suportada.

Exemplos: Geometria, Raster -- resultando em geometria vals

```
SELECT
    foo.rid,
    foo.gid,
    ST_AsText((foo.geomval).geom) As geomwkt,
    (foo.geomval).val
FROM (
    SELECT
        A.rid,
        g.gid,
        ST_Intersection(A.rast, g.geom) As geomval
    FROM dummy_rast AS A
    CROSS JOIN (
        VALUES
            (1, ST_Point(3427928, 5793243.85) ),
            (2, ST_GeomFromText('LINESTRING(3427927.85 5793243.75,3427927.8  ↩
               5793243.75,3427927.8 5793243.8)')),
            (3, ST_GeomFromText('LINESTRING(1 2, 3 4)'))
    ) As g(gid,geom)
    WHERE A.rid = 2
) As foo;

 rid | gid |      geomwkt                                                                                  | val
-----+-----+-----------------------------------------------------------------------------------------------

   2 |   1 | POINT(3427928 5793243.85)                                                                     | 249
   2 |   1 | POINT(3427928 5793243.85)                                                                     | 253
   2 |   2 | POINT(3427927.85 5793243.75)                                                                  | 254
   2 |   2 | POINT(3427927.8 5793243.8)                                                                    | 251
   2 |   2 | POINT(3427927.8 5793243.8)                                                                    | 253
   2 |   2 | LINESTRING(3427927.8 5793243.75,3427927.8 5793243.8)    | 252
   2 |   2 | MULTILINESTRING((3427927.8 5793243.8,3427927.8 5793243.75),...) | 250
   2 |   3 | GEOMETRYCOLLECTION EMPTY
```

Veja também

geomval, ST_Intersects, ST_MapAlgebraExpr, ST_Clip, ST_AsText

10.12.5 Funções retorno de mapa algébrico embutido

Funções retorno de mapa algébrico embutido — Versão função retorno - Retorna um raster de uma banda dado um ou mais rasters de entrada, os índices e uma função retorno de um usuário específico.

Synopsis

raster **ST_MapAlgebra**(rastbandarg[] rastbandargset, regprocedure callbackfunc, text pixeltype=NULL, text extenttype=INTERSECTIO
raster customextent=NULL, integer distancex=0, integer distancey=0, text[] VARIADIC userargs=NULL);
raster **ST_MapAlgebra**(raster rast, integer[] nband, regprocedure callbackfunc, text pixeltype=NULL, text extenttype=FIRST,
raster customextent=NULL, integer distancex=0, integer distancey=0, text[] VARIADIC userargs=NULL);
raster **ST_MapAlgebra**(raster rast, integer nband, regprocedure callbackfunc, text pixeltype=NULL, text extenttype=FIRST,
raster customextent=NULL, integer distancex=0, integer distancey=0, text[] VARIADIC userargs=NULL);
raster **ST_MapAlgebra**(raster rast1, integer nband1, raster rast2, integer nband2, regprocedure callbackfunc, text pixeltype=NULL,
text extenttype=INTERSECTION, raster customextent=NULL, integer distancex=0, integer distancey=0, text[] VARIADIC userargs=NULL);
raster **ST_MapAlgebra**(raster rast, integer nband, regprocedure callbackfunc, float8[] mask, boolean weighted, text pixeltype=NULL, text extenttype=INTERSECTION, raster customextent=NULL, text[] VARIADIC userargs=NULL);

Descrição

Retorna um raster de uma banda dado um ou mais rasters de entrada, os índices e uma função retorno de um usuário específico.

rast,rast1,rast2, rastbandargset Rasters onde o processo do mapa algébrico é avaliado.

`rastbandargset` permite o uso de uma operação do mapa algébrico em vários rasters e/ou bandas. Veja o exemplo da Variante 1.

nband, nband1, nband2 Os números de banda do raster a ser avaliado. nband pode ser um inteiro ou inteiro [] indicando as bandas. nband1 é uma banda no rast1 e nband2 é banda no rast2 para caso hte 2 raster/2band.

callbackfunc The `callbackfunc` parameter must be the name and signature of an SQL or PL/pgSQL function, cast to a regprocedure. An example PL/pgSQL function example is:

```
CREATE OR REPLACE FUNCTION sample_callbackfunc(value double precision[][][], position ↩
    integer[][], VARIADIC userargs text[])
    RETURNS double precision
    AS $$
    BEGIN
        RETURN 0;
    END;
    $$ LANGUAGE 'plpgsql' IMMUTABLE;
```

The `callbackfunc` must have three arguments: a 3-dimension double precision array, a 2-dimension integer array and a variadic 1-dimension text array. The first argument `value` is the set of values (as double precision) from all input rasters. The three dimensions (where indexes are 1-based) are: raster #, row y, column x. The second argument `position` is the set of pixel positions from the output raster and input rasters. The outer dimension (where indexes are 0-based) is the raster #. The position at outer dimension index 0 is the output raster's pixel position. For each outer dimension, there are two elements in the inner dimension for X and Y. The third argument `userargs` is for passing through any user-specified arguments.

Passing a regprocedure argument to a SQL function requires the full function signature to be passed, then cast to a regprocedure type. To pass the above example PL/pgSQL function as an argument, the SQL for the argument is:

```
'sample_callbackfunc(double precision[], integer[], text[])'::regprocedure
```

Note that the argument contains the name of the function, the types of the function arguments, quotes around the name and argument types, and a cast to a regprocedure.

mask An n-dimensional array (matrix) of numbers used to filter what cells get passed to map algebra call-back function. 0 means a neighbor cell value should be treated as no-data and 1 means value should be treated as data. If weight is set to true, then the values, are used as multipliers to multiple the pixel value of that value in the neighborhood position.

weighted booleano (verdadeiro/falso) para indicar se o valor da máscara deveria ser pesado (multiplicado pelo valor original) ou não (só se aplica para protocolo que usa máscara).

pixeltype Se um `pixeltype` for passado, a banda do novo raster será desse tipo de pixel. Se ele passar NULO ou for deixado, a nova banda raster terá o mesmo tipo de pixel da banda especificada do primeiro raster (para tipos de extensão: INTERSEÇÃO, UNIÃO, PRIMEIRO, CUSTOM) ou da banda específica do raster apropriado (para tipos de extensão: SEGUNDO, ÚLTIMO). Se estiver em dúvida, sempre especifique `pixeltype`.

O tipo de pixel resultante do raster de saída devem ser listados em ST_BandPixelType ou deixado de fora ou NULO.

extenttype Possíveis valores são INTERSEÇÃO (padrão), UNIÃO, PRIMEIRO (padrão para uma variante raster), SEGUNDO, ÚLTIMO, CUSTOM.

customextent Se `extentype` for CUSTOM, um raster deve ser fornecido para `customextent`. Veja o exemplo 4 de Variante 1.

distancex The distance in pixels from the reference cell in x direction. So width of resulting matrix would be `2*distancex + 1`.If not specified only the reference cell is considered (neighborhood of 0).

distancey A distância em pixeis da célula de referência na direção y. A altura da matriz resultante seria `2*distancey + 1`. Se não especificada, apenas a célula referência é considerada (vizinhança de 0).

userargs O terceiro argumento para a `callbackfunc` é um arranjo variadic text. Todos os argumentos de caminho de texto são passados pelo `callbackfunc` especificado, e são contados no argumento `userargs`.

Note
Para maiores informações sobre a palavra-chave VARIADIC, por favor recorra à documentação do PostgreSQL e a seção "SQL Functions with Variable Numbers of Arguments" do Query Language (SQL) Functions.

Note
O argumento text[] para a `callbackfunc` é requerido, independente de onde você escolher passar qualquer argumento para a função retorno para processar ou não.

A Variante 1 aceita um arranjo de `rastbandarg` permitindo o uso da operação de mapa algébrico em vários rasters e/ou bandas. Veja o exemplo de Variante 1.

As Variantes 2 e 3 operam em uma ou mais bandas de um raster. Veja os exemplos das Variantes 2 e 3.

A Variante 4 opera em dois raster com uma banda por raster. Veja o exemplo da Variante 4.

Disponibilidade: 2.2.0: Habilidade de adicionar máscara

Disponibilidade: 2.1.0

Exemplos: Variante 1

Um raster, uma banda

```
WITH foo AS (
    SELECT 1 AS rid, ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, -1, 0, 0, 0), 1, '16BUI', ↩
        1, 0) AS rast
)
SELECT
    ST_MapAlgebra(
        ARRAY[ROW(rast, 1)]::rastbandarg[],
        'sample_callbackfunc(double precision[], int[], text[])'::regprocedure
    ) AS rast
FROM foo
```

Um raster, várias bandas

```
WITH foo AS (
    SELECT 1 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, -1, ↩
         0, 0, 0), 1, '16BUI', 1, 0), 2, '8BUI', 10, 0), 3, '32BUI', 100, 0) AS rast
)
SELECT
    ST_MapAlgebra(
        ARRAY[ROW(rast, 3), ROW(rast, 1), ROW(rast, 3), ROW(rast, 2)]::rastbandarg[],
        'sample_callbackfunc(double precision[], int[], text[])'::regprocedure
    ) AS rast
FROM foo
```

Vários rasters, várias bandas

```
WITH foo AS (
    SELECT 1 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, -1, ↩
         0, 0, 0), 1, '16BUI', 1, 0), 2, '8BUI', 10, 0), 3, '32BUI', 100, 0) AS rast UNION  ↩
        ALL
    SELECT 2 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 1, 1, -1, ↩
         0, 0, 0), 1, '16BUI', 2, 0), 2, '8BUI', 20, 0), 3, '32BUI', 300, 0) AS rast
)
SELECT
    ST_MapAlgebra(
        ARRAY[ROW(t1.rast, 3), ROW(t2.rast, 1), ROW(t2.rast, 3), ROW(t1.rast, 2)]:: ↩
            rastbandarg[],
        'sample_callbackfunc(double precision[], int[], text[])'::regprocedure
    ) AS rast
FROM foo t1
CROSS JOIN foo t2
WHERE t1.rid = 1
    AND t2.rid = 2
```

Exemplo completo de tiles de uma cobertura com vizinhança. Esta consulta funciona apenas com PostgreSQL 9.1 ou superior.

```
WITH foo AS (
    SELECT 0 AS rid, ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, -1, 0, 0, 0), 1, '16BUI', ↩
         1, 0) AS rast UNION ALL
    SELECT 1, ST_AddBand(ST_MakeEmptyRaster(2, 2, 2, 0, 1, -1, 0, 0, 0), 1, '16BUI', 2, 0)  ↩
        AS rast UNION ALL
    SELECT 2, ST_AddBand(ST_MakeEmptyRaster(2, 2, 4, 0, 1, -1, 0, 0, 0), 1, '16BUI', 3, 0)  ↩
        AS rast UNION ALL

    SELECT 3, ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, -2, 1, -1, 0, 0, 0), 1, '16BUI', 10,  ↩
        0) AS rast UNION ALL
    SELECT 4, ST_AddBand(ST_MakeEmptyRaster(2, 2, 2, -2, 1, -1, 0, 0, 0), 1, '16BUI', 20,  ↩
        0) AS rast UNION ALL
    SELECT 5, ST_AddBand(ST_MakeEmptyRaster(2, 2, 4, -2, 1, -1, 0, 0, 0), 1, '16BUI', 30,  ↩
        0) AS rast UNION ALL

    SELECT 6, ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, -4, 1, -1, 0, 0, 0), 1, '16BUI', 100,  ↩
        0) AS rast UNION ALL
    SELECT 7, ST_AddBand(ST_MakeEmptyRaster(2, 2, 2, -4, 1, -1, 0, 0, 0), 1, '16BUI', 200,  ↩
        0) AS rast UNION ALL
    SELECT 8, ST_AddBand(ST_MakeEmptyRaster(2, 2, 4, -4, 1, -1, 0, 0, 0), 1, '16BUI', 300,  ↩
        0) AS rast
)
SELECT
    t1.rid,
    ST_MapAlgebra(
        ARRAY[ROW(ST_Union(t2.rast), 1)]::rastbandarg[],
        'sample_callbackfunc(double precision[], int[], text[])'::regprocedure,
        '32BUI',
```

```
        'CUSTOM', t1.rast,
        1, 1
    ) AS rast
FROM foo t1
CROSS JOIN foo t2
WHERE t1.rid = 4
    AND t2.rid BETWEEN 0 AND 8
    AND ST_Intersects(t1.rast, t2.rast)
GROUP BY t1.rid, t1.rast
```

Exemplo como o anterior, mas funciona com o PostgreSQL 9.0.

```
WITH src AS (
    SELECT 0 AS rid, ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, -1, 0, 0, 0), 1, '16BUI', ↩
         1, 0) AS rast UNION ALL
    SELECT 1, ST_AddBand(ST_MakeEmptyRaster(2, 2, 2, 0, 1, -1, 0, 0, 0), 1, '16BUI', 2, 0)  ↩
        AS rast UNION ALL
    SELECT 2, ST_AddBand(ST_MakeEmptyRaster(2, 2, 4, 0, 1, -1, 0, 0, 0), 1, '16BUI', 3, 0)  ↩
        AS rast UNION ALL

    SELECT 3, ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, -2, 1, -1, 0, 0, 0), 1, '16BUI', 10,  ↩
        0) AS rast UNION ALL
    SELECT 4, ST_AddBand(ST_MakeEmptyRaster(2, 2, 2, -2, 1, -1, 0, 0, 0), 1, '16BUI', 20,  ↩
        0) AS rast UNION ALL
    SELECT 5, ST_AddBand(ST_MakeEmptyRaster(2, 2, 4, -2, 1, -1, 0, 0, 0), 1, '16BUI', 30,  ↩
        0) AS rast UNION ALL

    SELECT 6, ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, -4, 1, -1, 0, 0, 0), 1, '16BUI', 100,  ↩
        0) AS rast UNION ALL
    SELECT 7, ST_AddBand(ST_MakeEmptyRaster(2, 2, 2, -4, 1, -1, 0, 0, 0), 1, '16BUI', 200,  ↩
        0) AS rast UNION ALL
    SELECT 8, ST_AddBand(ST_MakeEmptyRaster(2, 2, 4, -4, 1, -1, 0, 0, 0), 1, '16BUI', 300,  ↩
        0) AS rast
)
WITH foo AS (
    SELECT
        t1.rid,
        ST_Union(t2.rast) AS rast
    FROM src t1
    JOIN src t2
        ON ST_Intersects(t1.rast, t2.rast)
        AND t2.rid BETWEEN 0 AND 8
    WHERE t1.rid = 4
    GROUP BY t1.rid
), bar AS (
    SELECT
        t1.rid,
        ST_MapAlgebra(
            ARRAY[ROW(t2.rast, 1)]::rastbandarg[],
            'raster_nmapalgebra_test(double precision[], int[], text[])'::regprocedure,
            '32BUI',
            'CUSTOM', t1.rast,
            1, 1
        ) AS rast
    FROM src t1
    JOIN foo t2
        ON t1.rid = t2.rid
)
SELECT
    rid,
    (ST_Metadata(rast)),
    (ST_BandMetadata(rast, 1)),
```

```
    ST_Value(rast, 1, 1, 1)
FROM bar;
```

Exemplos: Variantes 2 e 3

Um raster, várias bandas

```
WITH foo AS (
    SELECT 1 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, -1, ↩
         0, 0, 0), 1, '16BUI', 1, 0), 2, '8BUI', 10, 0), 3, '32BUI', 100, 0) AS rast
)
SELECT
    ST_MapAlgebra(
        rast, ARRAY[3, 1, 3, 2]::integer[],
        'sample_callbackfunc(double precision[], int[], text[])'::regprocedure
    ) AS rast
FROM foo
```

Um raster, uma banda

```
WITH foo AS (
    SELECT 1 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, -1, ↩
         0, 0, 0), 1, '16BUI', 1, 0), 2, '8BUI', 10, 0), 3, '32BUI', 100, 0) AS rast
)
SELECT
    ST_MapAlgebra(
        rast, 2,
        'sample_callbackfunc(double precision[], int[], text[])'::regprocedure
    ) AS rast
FROM foo
```

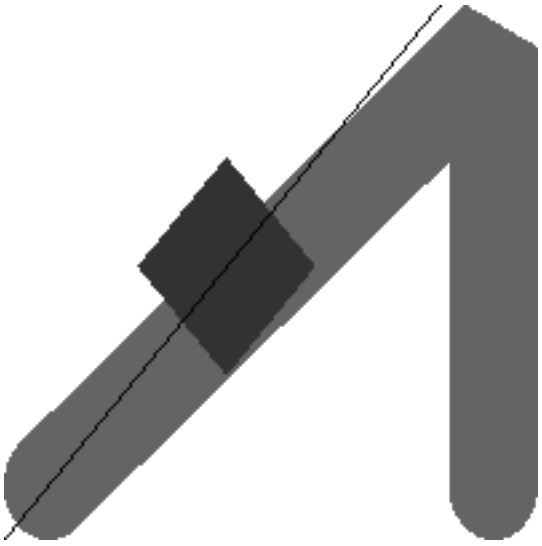
Exemplos: Variante 4

Dois rasters, duas bandas

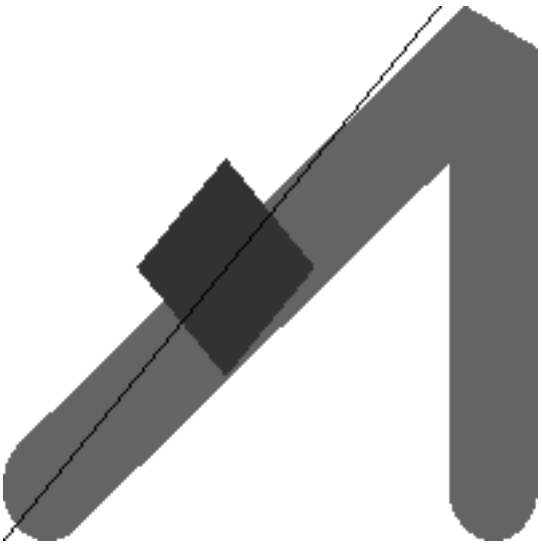
```
WITH foo AS (
    SELECT 1 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, -1, ↩
         0, 0, 0), 1, '16BUI', 1, 0), 2, '8BUI', 10, 0), 3, '32BUI', 100, 0) AS rast UNION  ↩
        ALL
    SELECT 2 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 1, 1, -1, ↩
         0, 0, 0), 1, '16BUI', 2, 0), 2, '8BUI', 20, 0), 3, '32BUI', 300, 0) AS rast
)
SELECT
    ST_MapAlgebra(
        t1.rast, 2,
        t2.rast, 1,
        'sample_callbackfunc(double precision[], int[], text[])'::regprocedure
    ) AS rast
FROM foo t1
CROSS JOIN foo t2
WHERE t1.rid = 1
    AND t2.rid = 2
```

Exemplos: Utilizando Máscaras

```
WITH foo AS (SELECT
   ST_SetBandNoDataValue(
ST_SetValue(ST_SetValue(ST_AsRaster(
        ST_Buffer(
            ST_GeomFromText('LINESTRING(50 50,100 90,100 50)'), 5,'join=bevel'),
            200,200,ARRAY['8BUI'], ARRAY[100], ARRAY[0]), ST_Buffer('POINT(70 70)':: ↩
                geometry,10,'quad_segs=1') ,50),
  'LINESTRING(20 20, 100 100, 150 98)'::geometry,1),0)  AS rast )
SELECT 'original' AS title, rast
FROM foo
UNION ALL
SELECT 'no mask mean value' AS title, ST_MapAlgebra(rast,1,'ST_mean4ma(double precision[],  ↩
    int[], text[])'::regprocedure) AS rast
FROM foo
UNION ALL
SELECT 'mask only consider neighbors, exclude center' AS title, ST_MapAlgebra(rast,1,' ↩
    ST_mean4ma(double precision[], int[], text[])'::regprocedure,
    '{{1,1,1}, {1,0,1}, {1,1,1}}'::double precision[], false) As rast
FROM foo

UNION ALL
SELECT 'mask weighted only consider neighbors, exclude center multi otehr pixel values by  ↩
    2' AS title, ST_MapAlgebra(rast,1,'ST_mean4ma(double precision[], int[], text[])':: ↩
    regprocedure,
    '{{2,2,2}, {2,0,2}, {2,2,2}}'::double precision[], true) As rast
FROM foo;
```

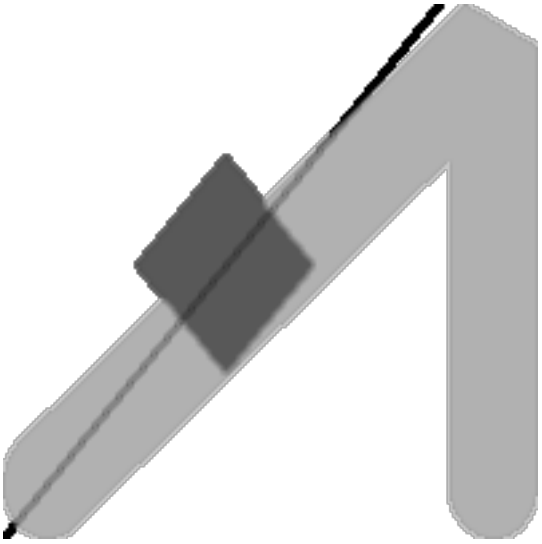


original



nenhuma máscara significa valor (é o mesmo que todos 1s na matriz máscara)

máscara considera somente vizinhos, exclui centro



máscara pesada considera somente vizinhos, exclui centro multi outros valores de pixel por 2

Veja também

rastbandarg, ST_Union, ST_MapAlgebraExpr

10.12.6 ST_MapAlgebraExpr

ST_MapAlgebraExpr — Versão expressão - Retorna um raster de uma banda dado um ou mais rasters de entrada, índices de banda e uma ou mais expressões SQL de usuários específicos.

Synopsis

raster **ST_MapAlgebra**(raster rast, integer nband, text pixeltype, text expression, double precision nodataval=NULL);
raster **ST_MapAlgebra**(raster rast, text pixeltype, text expression, double precision nodataval=NULL);
raster **ST_MapAlgebra**(raster rast1, integer nband1, raster rast2, integer nband2, text expression, text pixeltype=NULL, text extenttype=INTERSECTION, text nodata1expr=NULL, text nodata2expr=NULL, double precision nodatanodataval=NULL);
raster **ST_MapAlgebra**(raster rast1, raster rast2, text expression, text pixeltype=NULL, text extenttype=INTERSECTION, text nodata1expr=NULL, text nodata2expr=NULL, double precision nodatanodataval=NULL);

Descrição

Versão expressão - Retorna um raster de uma banda dado um ou mais rasters de entrada, índices de banda e uma ou mais expressões SQL de usuários específicos.

Disponibilidade: 2.1.0

Descrição: Variantes 1 e 2 (um raster)

Cria uma nova banda raster formada pela aplicação válida de uma operação algébrica PostgreSQL definida pela `expression` no raster de saída (`rast`). Se `nband` não for dado, a banda 1 é assumida. O novo raster terá a mesma georreferência, largura e altura que o raster original, mas só terá uma banda.

Se um `pixeltype` passar, então o novo raster terá a mesma banda dele. Se o tipo de pixel passar NULO, a nova banda raster terá o mesmo tipo de pixel que a banda de entrada `rast`.

- Palavras-chave permitidas para `expression`

  1. `[rast]` - Valor do pixel de interesse
  2. `[rast.val]` - Valor do pixel de interesse
  3. `[rast.x]` - coluna pixel 1-baseada do pixel de interesse
  4. `[rast.y]` - linha pixel 1-baseada do pixel de interesse

Descrição: Variantes 3 e 4 (dois rasters)

Cria uma nova banda raster formada pela aplicação válida de uma operação algébrica PostgreSQL definida pela `expression` no raster de saída (`rast`). Se `nband`, `band2` não forem especificados, a banda 1 é assumida. O raster resultante será alinhado (escala, inclinação e cantos de pixel) na grade definida pelo primeiro raster. O raster resultante terá de ser definido pelo primeiro raster. O raster resultante terá a extensão definida pelo parâmetro `extenttype`.

expressão Uma expressão algébrica PostgreSQL envolvendo dois rasters e funções/operadores PostgreSQL definidos que irão elucidar o valor do pixel quando eles se intersectarem. ex.: (([rast1] + [rast2])/2.0)::integer

pixeltype O tipo de pixel resultante do raster de saída. Deve ser um listado em ST_BandPixelType, deixado de fora ou NULO. Se não passar ou for NULO, usa-se o tipo de pixel do primeiro raster.

extenttype Controla a extensão do raster resultante

1. `INTERSECTION` - A extensão do novo raster é a interseção de dois rasters. Este é o padrão.
2. `UNION` - A extensão do novo raster é a união dos dois raster.
3. `FIRST` - A extensão do novo raster é a mesma da do primeiro raster.
4. `SECOND` - A extensão do novo raster é a mesma da do segundo raster.

nodata1expr Uma expressão algébrica envolvendo somente `rast2` ou uma constante que define o que retornar quando pixeis de `rast1` são valores nodata e os pixeis rast2 têm valores.

nodata2expr Uma expressão algébrica envolvendo somente `rast1` ou uma constante que define o que retornar quando pixeis de `rast2` são valores nodata e os pixeis rast1 têm valores.

nodatanodataval Uma constante numérica para retornar quando os pixeis rast1 e raste2 forem ambos valores nodata.

- Palavras-chave permitidas em `expression`, `nodata1expr` e `nodata2expr`

  1. `[rast1]` - Valor do pixel de interesse do `rast1`
  2. `[rast1.val]` - Valor do pixel de interesse do `rast1`
  3. `[rast1.x]` - coluna pixel 1-based do pixel de interesse do`rast1`
  4. `[rast1.y]` - linha pixel 1-based do pixel de interesse do`rast1`
  5. `[rast2]` - Valor do pixel de interesse do `rast2`
  6. `[rast2.val]` - Valor do pixel de interesse do `rast2`
  7. `[rast2.x]` - coluna pixel 1-based do pixel de interesse do`rast2`
  8. `[rast2.y]` - linha pixel 1-based do pixel de interesse do`rast2`

Exemplos: Variantes 1 e 2

```
WITH foo AS (
    SELECT ST_AddBand(ST_MakeEmptyRaster(10, 10, 0, 0, 1, 1, 0, 0, 0), '32BF'::text, 1, -1) ↩
        AS rast
)
SELECT
    ST_MapAlgebra(rast, 1, NULL, 'ceil([rast]*[rast.x]/[rast.y]+[rast.val])')
FROM foo;
```

Exemplos: Variantes 3 e 4

```
WITH foo AS (
    SELECT 1 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 0, 1, -1, ↩
         0, 0, 0), 1, '16BUI', 1, 0), 2, '8BUI', 10, 0), 3, '32BUI'::text, 100, 0) AS rast  ↩
        UNION ALL
    SELECT 2 AS rid, ST_AddBand(ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(2, 2, 0, 1, 1, -1, ↩
         0, 0, 0), 1, '16BUI', 2, 0), 2, '8BUI', 20, 0), 3, '32BUI'::text, 300, 0) AS rast
)
SELECT
    ST_MapAlgebra(
        t1.rast, 2,
        t2.rast, 1,
        '([rast2] + [rast1.val]) / 2'
    ) AS rast
FROM foo t1
CROSS JOIN foo t2
WHERE t1.rid = 1
    AND t2.rid = 2;
```

Veja também

rastbandarg, ST_Union, Funções retorno de mapa algébrico embutido

10.12.7 ST_MapAlgebraExpr

ST_MapAlgebraExpr — Versão de banda raster 1: Cria uma nova banda raster formada pela aplicação de ma operação algébrica válida do PostgreSQL na banda raster de entrada de um tipo de pixel fornecido. A banda 1 é assumida se nenhuma banda for especificada.

Synopsis

raster **ST_MapAlgebraExpr**(raster rast, integer band, text pixeltype, text expression, double precision nodataval=NULL);
raster **ST_MapAlgebraExpr**(raster rast, text pixeltype, text expression, double precision nodataval=NULL);

Descrição

Warning
ST_MapAlgebraExpr é menosprezado como do 2.1.0. Use ST_MapAlgebraExpr.

Cria uma nova banda raster formada pela aplicação válida de uma operação algébrica PostgreSQL definida pela `expression` no raster de entrada (`rast`). Se `band` não for dado, a banda 1 é assumida. O novo raster terá a mesma georreferência, largura e altura que o raster original, mas só terá uma banda.

Se um `pixeltype` passar, então o novo raster terá a mesma banda dele. Se o tipo de pixel passar NULO, a nova banda raster terá o mesmo tipo de pixel que a banda de entrada `rast`.

Na expressão você pode usar o termo `[rast]` para referir o valor do pixel da banda original, `[rast.x]` para referir ao índice da coluna pixel 1-baseada, `[rast.y]` para referir ao índice da linha pixel 1-baseada.

Disponibilidade: 2.0.0

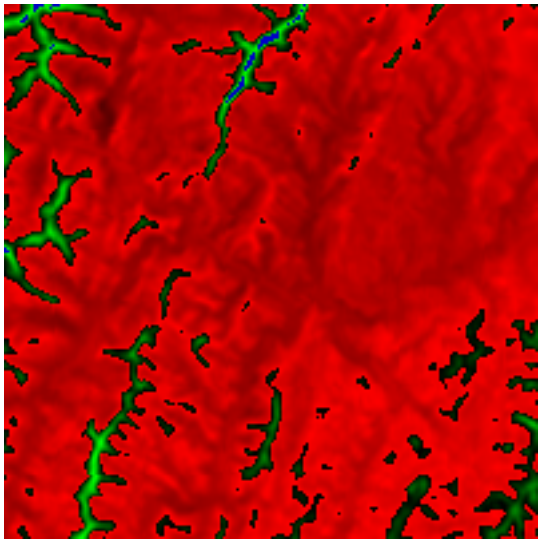
Exemplos

Cria uma nova banda raster 1 a partir da nossa original que é uma função de módulo 2 da banda raster original.

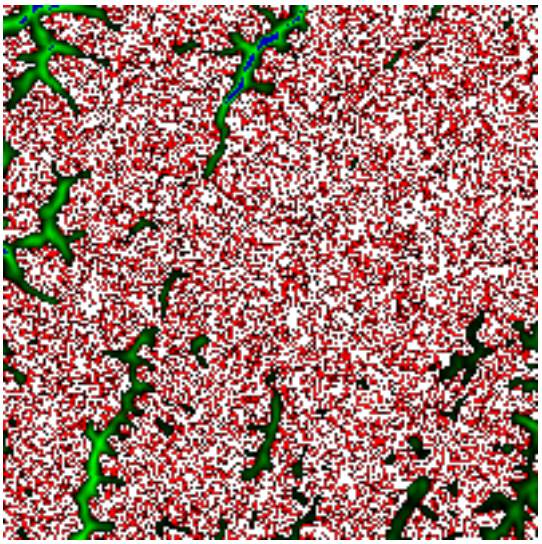
```
ALTER TABLE dummy_rast ADD COLUMN map_rast raster;
UPDATE dummy_rast SET map_rast = ST_MapAlgebraExpr(rast,NULL,'mod([rast]::numeric,2)')  ↩
    WHERE rid = 2;

SELECT
    ST_Value(rast,1,i,j) As origval,
    ST_Value(map_rast, 1, i, j) As mapval
FROM dummy_rast
CROSS JOIN generate_series(1, 3) AS i
CROSS JOIN generate_series(1,3) AS j
WHERE rid = 2;

 origval | mapval
---------+--------
     253 |      1
     254 |      0
     253 |      1
     253 |      1
     254 |      0
     254 |      0
     250 |      0
     254 |      0
     254 |      0
```

Cria uma nova banda raster 1 do tipo de pixel 2BUI a partir da nossa original que é reclassificada e obtém valor nodata 0.

```
ALTER TABLE dummy_rast ADD COLUMN map_rast2 raster;
UPDATE dummy_rast SET
    map_rast2 = ST_MapAlgebraExpr(rast,'2BUI'::text,'CASE WHEN [rast] BETWEEN 100 and 250  ↩
        THEN 1 WHEN [rast] = 252 THEN 2 WHEN [rast] BETWEEN 253 and 254 THEN 3 ELSE 0 END':: ↩
        text, '0')
WHERE rid = 2;

SELECT DISTINCT
    ST_Value(rast,1,i,j) As origval,
    ST_Value(map_rast2, 1, i, j) As mapval
FROM dummy_rast
CROSS JOIN generate_series(1, 5) AS i
CROSS JOIN generate_series(1,5) AS j
WHERE rid = 2;

 origval | mapval
---------+--------
     249 |      1
     250 |      1
     251 |
     252 |      2
     253 |      3
     254 |      3

SELECT
    ST_BandPixelType(map_rast2) As b1pixtyp
FROM dummy_rast
WHERE rid = 2;

 b1pixtyp
----------
 2BUI
```



original (column rast_view)



rast_view_ma

Cria uma nova banda raster 3 do mesmo tipo de pixel da nossa banda 3 original, com a primeira banda alterada pelo mapa algébrico e 2 bandas permanecem inalteradas.

```
SELECT
    ST_AddBand(
        ST_AddBand(
            ST_AddBand(
                ST_MakeEmptyRaster(rast_view),
                ST_MapAlgebraExpr(rast_view,1,NULL,'tan([rast])*[rast]')
            ),
            ST_Band(rast_view,2)
        ),
        ST_Band(rast_view, 3)
    )  As rast_view_ma
FROM wind
WHERE rid=167;
```

Veja também

ST_MapAlgebraExpr, ST_MapAlgebraFct, ST_BandPixelType, ST_GeoReference, ST_Value

10.12.8 ST_MapAlgebraExpr

ST_MapAlgebraExpr — Versão de banda raster 2: Cria uma banda raster nova formada pela aplicação de uma operação algébrica válida PostgreSQL nas duas bandas raster de entrada e do tipo de pixel fornecido. A banda 1 de cada raster é assumida se nenhum número de bandas for especificado. O raster resultante será alinhado (escala, inclinação e cantos de pixel) na grade definida pelo primeiro raster e tem sua extensão definida pelo parâmetro "extenttype". O valores para "extenttype" pode ser: INTERSEÇÃO, UNIÃO, PRIMEIRO, SEGUNDO.

Synopsis

raster **ST_MapAlgebraExpr**(raster rast1, raster rast2, text expression, text pixeltype=same_as_rast1_band, text extenttype=INTERSEC
text nodata1expr=NULL, text nodata2expr=NULL, double precision nodatanodataval=NULL);

raster **ST_MapAlgebraExpr**(raster rast1, integer band1, raster rast2, integer band2, text expression, text pixeltype=same_as_rast1_band text extenttype=INTERSECTION, text nodata1expr=NULL, text nodata2expr=NULL, double precision nodatanodataval=NULL);

Descrição

Warning
ST_MapAlgebraExpr é menosprezado como do 2.1.0. Use ST_MapAlgebraExpr.

Cria uma nova banda raster formada pela aplicação válida de uma operação algébrica PostgreSQL definida pela `expression` no raster de saída (`rast`). Se `nband`, `band2` não forem especificados, a banda 1 é assumida. O raster resultante será alinhado (escala, inclinação e cantos de pixel) na grade definida pelo primeiro raster. O raster resultante terá de ser definido pelo primeiro raster. O raster resultante terá a extensão definida pelo parâmetro `extenttype`.

expressão Uma expressão algébrica PostgreSQL envolvendo dois rasters e funções/operadores PostgreSQL definidos que irão elucidar o valor do pixel quando eles se intersectarem. ex.: (([rast1] + [rast2])/2.0)::integer

pixeltype O tipo de pixel resultante do raster de saída. Deve ser um listado em ST_BandPixelType, deixado de fora ou NULO. Se não passar ou for NULO, usa-se o tipo de pixel do primeiro raster.

extenttype Controla a extensão do raster resultante

1. `INTERSECTION` - A extensão do novo raster é a interseção de dois rasters. Este é o padrão.
2. `UNION` - A extensão do novo raster é a união dos dois raster.
3. `FIRST` - A extensão do novo raster é a mesma da do primeiro raster.
4. `SECOND` - A extensão do novo raster é a mesma da do segundo raster.

nodata1expr Uma expressão algébrica envolvendo somente `rast2` ou uma constante que define o que retornar quando pixeis de `rast1` são valores nodata e os pixeis rast2 têm valores.

nodata2expr Uma expressão algébrica envolvendo somente `rast1` ou uma constante que define o que retornar quando pixeis de `rast2` são valores nodata e os pixeis rast1 têm valores.

nodatanodataval Uma constante numérica para retornar quando os pixeis rast1 e raste2 forem ambos valores nodata.

Se `pixeltype` passar, o novo raster terá uma banda desse tipo de pixel. Se o tipo de pixel passar NULOou nenhum tipo for especificado, a nova banda raster terá o mesmo tipo de pixel da banda de entrada `rast1`.

Use o termo `[rast1.val] [rast2.val]` para referir-se ao valor de pixel das bandas rasters originais e `[rast1.x]`, `[rast1.y]` etc. para referir-se à posição da coluna/linha dos pixeis.

Disponibilidade: 2.0.0

Exemplo: 2 Interseção de Banda e União

Cria uma nova banda raster 1 a partir da nossa original que é uma função de módulo 2 da banda raster original.

```
--Create a cool set of rasters --
DROP TABLE IF EXISTS fun_shapes;
CREATE TABLE fun_shapes(rid serial PRIMARY KEY, fun_name text, rast raster);

-- Insert some cool shapes around Boston in Massachusetts state plane meters --
INSERT INTO fun_shapes(fun_name, rast)
VALUES ('ref', ST_AsRaster(ST_MakeEnvelope(235229, 899970, 237229, 901930,26986),200,200,'8 ↩
    BUI',0,0));
```

```
INSERT INTO fun_shapes(fun_name,rast)
WITH ref(rast) AS (SELECT rast FROM fun_shapes WHERE fun_name = 'ref' )
SELECT 'area' AS fun_name, ST_AsRaster(ST_Buffer(ST_SetSRID(ST_Point(236229, 900930),26986) ↩
    , 1000),
            ref.rast,'8BUI', 10, 0) As rast
FROM ref
UNION ALL
SELECT 'rand bubbles',
            ST_AsRaster(
            (SELECT ST_Collect(geom)
    FROM (SELECT ST_Buffer(ST_SetSRID(ST_Point(236229 + i*random()*100, 900930 + j*random() ↩
        *100),26986), random()*20) As geom
            FROM generate_series(1,10) As i, generate_series(1,10) As j
            ) As foo ), ref.rast,'8BUI', 200, 0)
FROM ref;

--map them -
SELECT  ST_MapAlgebraExpr(
        area.rast, bub.rast, '[rast2.val]', '8BUI', 'INTERSECTION', '[rast2.val]', '[rast1. ↩
            val]') As interrast,
        ST_MapAlgebraExpr(
            area.rast, bub.rast, '[rast2.val]', '8BUI', 'UNION', '[rast2.val]', '[rast1.val ↩
                ]') As unionrast
FROM
  (SELECT rast FROM fun_shapes WHERE
 fun_name = 'area') As area
CROSS JOIN  (SELECT rast
FROM fun_shapes WHERE
 fun_name = 'rand bubbles') As bub
```

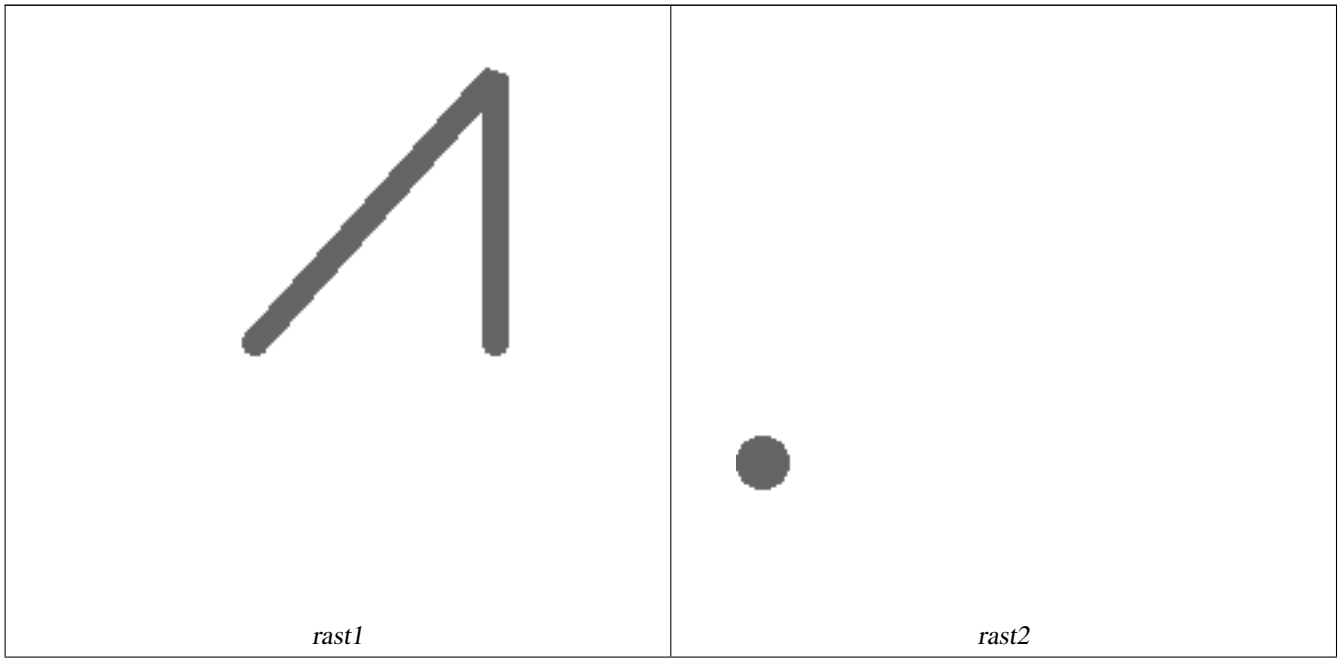
interseção de mapa algébrico

união de mapa algébrico

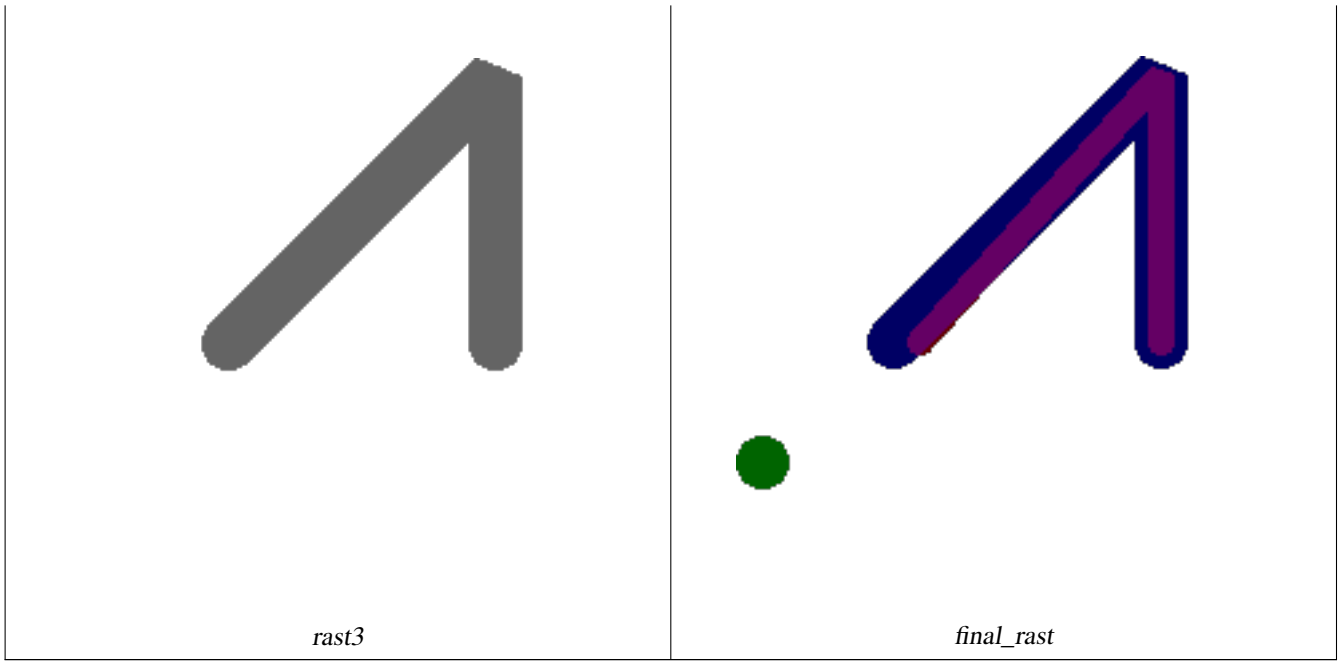
Exemplo: Revestindo rasters em um quadro como bandas separadas

```
-- we use ST_AsPNG to render the image so all single band ones look grey --
WITH mygeoms
    AS ( SELECT 2 As bnum, ST_Buffer(ST_Point(1,5),10) As geom
            UNION ALL
```

```
          SELECT 3 AS bnum,
              ST_Buffer(ST_GeomFromText('LINESTRING(50 50,150 150,150 50)'), 10,'join= ↩
                  bevel') As geom
          UNION ALL
          SELECT 1 As bnum,
              ST_Buffer(ST_GeomFromText('LINESTRING(60 50,150 150,150 50)'), 5,'join= ↩
                  bevel') As geom
          ),
-- define our canvas to be 1 to 1 pixel to geometry
canvas
 AS (SELECT ST_AddBand(ST_MakeEmptyRaster(200,
     200,
     ST_XMin(e)::integer, ST_YMax(e)::integer, 1, -1, 0, 0) , '8BUI'::text,0) As rast
     FROM (SELECT ST_Extent(geom) As e,
                 Max(ST_SRID(geom)) As srid
                 from mygeoms
                 ) As foo
          ),
rbands AS (SELECT ARRAY(SELECT ST_MapAlgebraExpr(canvas.rast, ST_AsRaster(m.geom, canvas ↩
    .rast, '8BUI', 100),
              '[rast2.val]', '8BUI', 'FIRST', '[rast2.val]', '[rast1.val]') As rast
            FROM mygeoms AS m CROSS JOIN canvas
            ORDER BY m.bnum) As rasts
            )
       SELECT rasts[1] As rast1 , rasts[2] As rast2, rasts[3] As rast3, ST_AddBand(
                 ST_AddBand(rasts[1],rasts[2]), rasts[3]) As final_rast
         FROM rbands;
```



rast1 *rast2*



rast3 | *final_rast*

Exemplo: Cobre 2 metros de limite das parcelas selecionadas sobre uma área imaginária

```
-- Create new 3 band raster composed of first 2 clipped bands, and overlay of 3rd band with ↩
     our geometry
-- This query took 3.6 seconds on PostGIS windows 64-bit install
WITH pr AS
-- Note the order of operation: we clip all the rasters to dimensions of our region
(SELECT ST_Clip(rast,ST_Expand(geom,50) ) As rast, g.geom
    FROM aerials.o_2_boston AS r INNER JOIN
-- union our parcels of interest so they form a single geometry we can later intersect with
        (SELECT ST_Union(ST_Transform(geom,26986)) AS geom
          FROM landparcels WHERE pid IN('0303890000', '0303900000')) As g
        ON ST_Intersects(rast::geometry, ST_Expand(g.geom,50))
),
-- we then union the raster shards together
-- ST_Union on raster is kinda of slow but much faster the smaller you can get the rasters
-- therefore we want to clip first and then union
prunion AS
(SELECT ST_AddBand(NULL, ARRAY[ST_Union(rast,1),ST_Union(rast,2),ST_Union(rast,3)] ) As  ↩
    clipped,geom
FROM pr
GROUP BY geom)
-- return our final raster which is the unioned shard with
-- with the overlay of our parcel boundaries
-- add first 2 bands, then mapalgebra of 3rd band + geometry
SELECT ST_AddBand(ST_Band(clipped,ARRAY[1,2])
    , ST_MapAlgebraExpr(ST_Band(clipped,3), ST_AsRaster(ST_Buffer(ST_Boundary(geom),2), ↩
       clipped, '8BUI',250),
     '[rast2.val]', '8BUI', 'FIRST', '[rast2.val]', '[rast1.val]') ) As rast
FROM prunion;
```

As linhas azuis são os limites das parcelas selecionadas

Veja também

ST_MapAlgebraExpr, ST_AddBand, ST_AsPNG, ST_AsRaster, ST_MapAlgebraFct, ST_BandPixelType, ST_GeoReference, ST_Value, ST_Union, ST_Union

10.12.9 ST_MapAlgebraFct

ST_MapAlgebraFct — Versão de banda raster 1: Cria uma nova banda raster formada pela aplicação de uma função válida do PostgreSQL na banda raster de entrada de um tipo de pixel fornecido. A banda 1 é assumida se nenhuma banda for especificada.

Synopsis

raster **ST_MapAlgebraFct**(raster rast, regprocedure onerasteruserfunc);
raster **ST_MapAlgebraFct**(raster rast, regprocedure onerasteruserfunc, text[] VARIADIC args);
raster **ST_MapAlgebraFct**(raster rast, text pixeltype, regprocedure onerasteruserfunc);
raster **ST_MapAlgebraFct**(raster rast, text pixeltype, regprocedure onerasteruserfunc, text[] VARIADIC args);
raster **ST_MapAlgebraFct**(raster rast, integer band, regprocedure onerasteruserfunc);
raster **ST_MapAlgebraFct**(raster rast, integer band, regprocedure onerasteruserfunc, text[] VARIADIC args);
raster **ST_MapAlgebraFct**(raster rast, integer band, text pixeltype, regprocedure onerasteruserfunc);
raster **ST_MapAlgebraFct**(raster rast, integer band, text pixeltype, regprocedure onerasteruserfunc, text[] VARIADIC args);

Descrição

Warning
ST_MapAlgebraFct é menosprezado como do 2.1.0. Use Funções retorno de mapa algébrico embutido.

Cria uma nova banda raster formada pela aplicação válida de uma função PostgreSQL definida pela `onerasteruserfunc` no raster de entrada (`rast`). Se `band` não for dado, a banda 1 é assumida. O novo raster terá a mesma georreferência, largura e altura que o raster original, mas só terá uma banda.

Se um `pixeltype` passar, então o novo raster terá a mesma banda dele. Se o tipo de pixel passar NULO, a nova banda raster terá o mesmo tipo de pixel que a banda de entrada `rast`.

The `onerasteruserfunc` parameter must be the name and signature of a SQL or PL/pgSQL function, cast to a regprocedure. A very simple and quite useless PL/pgSQL function example is:

```
CREATE OR REPLACE FUNCTION simple_function(pixel FLOAT, pos INTEGER[], VARIADIC args TEXT ↩
    [])
    RETURNS FLOAT
    AS $$ BEGIN
        RETURN 0.0;
    END; $$
    LANGUAGE 'plpgsql' IMMUTABLE;
```

The `userfunction` may accept two or three arguments: a float value, an optional integer array, and a variadic text array. The first argument is the value of an individual raster cell (regardless of the raster datatype). The second argument is the position of the current processing cell in the form '{x,y}'. The third argument indicates that all remaining parameters to ST_MapAlgebraFct shall be passed through to the `userfunction`.

Passing a regprodedure argument to a SQL function requires the full function signature to be passed, then cast to a regprocedure type. To pass the above example PL/pgSQL function as an argument, the SQL for the argument is:

```
'simple_function(float,integer[],text[])'::regprocedure
```

Note that the argument contains the name of the function, the types of the function arguments, quotes around the name and argument types, and a cast to a regprocedure.

O terceiro argumento para a `userfunction` é um variadic text arranjo. Todos os argumentos seguindo qualquer chamada ST_MapAlgebraFct passam pela `userfunction` especificada, e são contidos no argumento `args`.

Note
Para maiores informações sobre a palavra-chave VARIADIC, por favor recorra à documentação do PostgreSQL e a seção "SQL Functions with Variable Numbers of Arguments" do Query Language (SQL) Functions.

Note
O argumento text[] para o `userfunction` é requerido, independente se escolher passar argumentos para sua função usuário processar ou não.

Disponibilidade: 2.0.0

Exemplos

Cria uma nova banda raster 1 a partir da nossa original que é uma função de módulo 2 da banda raster original.

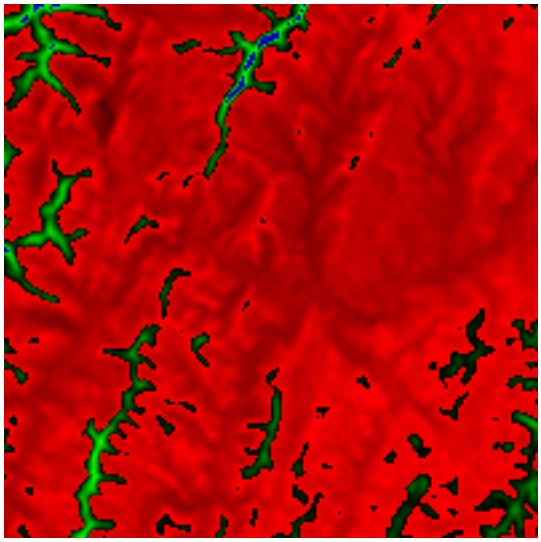
```
ALTER TABLE dummy_rast ADD COLUMN map_rast raster;
CREATE FUNCTION mod_fct(pixel float, pos integer[], variadic args text[])
RETURNS float
AS $$
BEGIN
    RETURN pixel::integer % 2;
END;
$$
```

```
LANGUAGE 'plpgsql' IMMUTABLE;

UPDATE dummy_rast SET map_rast = ST_MapAlgebraFct(rast,NULL,'mod_fct(float,integer[],text ↩
    [])'::regprocedure) WHERE rid = 2;

SELECT ST_Value(rast,1,i,j) As origval, ST_Value(map_rast, 1, i, j) As mapval
FROM dummy_rast CROSS JOIN generate_series(1, 3) AS i CROSS JOIN generate_series(1,3) AS j
WHERE rid = 2;

 origval | mapval
---------+--------
     253 |      1
     254 |      0
     253 |      1
     253 |      1
     254 |      0
     254 |      0
     250 |      0
     254 |      0
     254 |      0
```

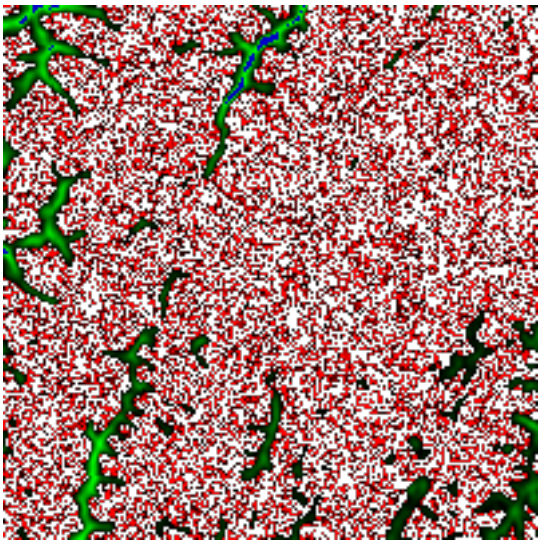
Cria uma nova banda raster 1 de tipo pixel 2BUI da original que é reclassificada e adquire valor sem dados para uma parâmetro passado à função usuário (0).

```
ALTER TABLE dummy_rast ADD COLUMN map_rast2 raster;
CREATE FUNCTION classify_fct(pixel float, pos integer[], variadic args text[])
RETURNS float
AS
$$
DECLARE
    nodata float := 0;
BEGIN
    IF NOT args[1] IS NULL THEN
        nodata := args[1];
    END IF;
    IF pixel < 251 THEN
        RETURN 1;
    ELSIF pixel = 252 THEN
        RETURN 2;
    ELSIF pixel > 252 THEN
        RETURN 3;
    ELSE
        RETURN nodata;
    END IF;
END;
$$
LANGUAGE 'plpgsql';
UPDATE dummy_rast SET map_rast2 = ST_MapAlgebraFct(rast,'2BUI','classify_fct(float,integer ↩
    [],text[])'::regprocedure, '0') WHERE rid = 2;

SELECT DISTINCT ST_Value(rast,1,i,j) As origval, ST_Value(map_rast2, 1, i, j) As mapval
FROM dummy_rast CROSS JOIN generate_series(1, 5) AS i CROSS JOIN generate_series(1,5) AS j
WHERE rid = 2;

 origval | mapval
---------+--------
     249 |      1
     250 |      1
     251 |
     252 |      2
     253 |      3
     254 |      3
```

```
SELECT ST_BandPixelType(map_rast2) As b1pixtyp
FROM dummy_rast WHERE rid = 2;

 b1pixtyp
----------
 2BUI
```



original (column rast-view)



rast_view_ma

Cria uma nova banda raster 3 do mesmo tipo de pixel da nossa banda 3 original, com a primeira banda alterada pelo mapa algébrico e 2 bandas permanecem inalteradas.

```
CREATE FUNCTION rast_plus_tan(pixel float, pos integer[], variadic args text[])
RETURNS float
AS
$$
BEGIN
    RETURN tan(pixel) * pixel;
END;
$$
LANGUAGE 'plpgsql';

SELECT ST_AddBand(
    ST_AddBand(
        ST_AddBand(
            ST_MakeEmptyRaster(rast_view),
            ST_MapAlgebraFct(rast_view,1,NULL,'rast_plus_tan(float,integer[],text[])':: ↩
                regprocedure)
        ),
        ST_Band(rast_view,2)
    ),
    ST_Band(rast_view, 3) As rast_view_ma
)
FROM wind
WHERE rid=167;
```

Veja também

ST_MapAlgebraExpr, ST_BandPixelType, ST_GeoReference, ST_SetValue

10.12.10 ST_MapAlgebraFct

ST_MapAlgebraFct — Versão de banda 2 - Cria uma nova banda raster um formada pela aplicação de uma função PostgreSQL na 2 entrada de bandas raster e do tipo de pixel fornecido. A banda 1 é assumida se nenhuma banda for especificada. Tipo de extensão torna-se INTERSEÇÃO se não especificada.

Synopsis

raster **ST_MapAlgebraFct**(raster rast1, raster rast2, regprocedure tworastuserfunc, text pixeltype=same_as_rast1, text extenttype=INTERSECTION, text[] VARIADIC userargs);
raster **ST_MapAlgebraFct**(raster rast1, integer band1, raster rast2, integer band2, regprocedure tworastuserfunc, text pixeltype=same_as_rast1, text extenttype=INTERSECTION, text[] VARIADIC userargs);

Descrição

Warning
ST_MapAlgebraFct é menosprezado como do 2.1.0. Use Funções retorno de mapa algébrico embutido.

Cria uma nova banda raster um formada pela aplicação válida de uma função PostgreSQL definida pela `tworastuserfunc` no raster de entrada `rast1`, `rast1`. Se `band1` ou `band2` não forem especificadas, a banda 1 é assumida. O novo raster terá a mesma georreferência, largura e altura que o raster original, mas só terá uma banda.

Se um `pixeltype` passar, então o novo raster terá a mesma banda dele. Se o tipo de pixel passar NULO, a nova banda raster terá o mesmo tipo de pixel que a banda de entrada `rast1`.

The `tworastuserfunc` parameter must be the name and signature of an SQL or PL/pgSQL function, cast to a regprocedure. An example PL/pgSQL function example is:

```
CREATE OR REPLACE FUNCTION simple_function_for_two_rasters(pixel1 FLOAT, pixel2 FLOAT, pos ↩
    INTEGER[], VARIADIC args TEXT[])
    RETURNS FLOAT
    AS $$ BEGIN
        RETURN 0.0;
    END; $$
    LANGUAGE 'plpgsql' IMMUTABLE;
```

The `tworastuserfunc` may accept three or four arguments: a double precision value, a double precision value, an optional integer array, and a variadic text array. The first argument is the value of an individual raster cell in `rast1` (regardless of the raster datatype). The second argument is an individual raster cell value in `rast2`. The third argument is the position of the current processing cell in the form '{x,y}'. The fourth argument indicates that all remaining parameters to ST_MapAlgebraFct shall be passed through to the `tworastuserfunc`.

Passing a regprodedure argument to a SQL function requires the full function signature to be passed, then cast to a regprocedure type. To pass the above example PL/pgSQL function as an argument, the SQL for the argument is:

```
'simple_function(double precision, double precision, integer[], text[])'::regprocedure
```

Note that the argument contains the name of the function, the types of the function arguments, quotes around the name and argument types, and a cast to a regprocedure.

The fourth argument to the `tworastuserfunc` is a variadic text array. All trailing text arguments to any ST_MapAlgebraFct call are passed through to the specified `tworastuserfunc`, and are contained in the `userargs` argument.

Note
Para maiores informações sobre a palavra-chave VARIADIC, por favor recorra à documentação do PostgreSQL e a seção "SQL Functions with Variable Numbers of Arguments" do Query Language (SQL) Functions.

Note
O argumento text[] para a `tworastuserfunc` é requerido, independente se escolher passar argumentos para sua função usuário processar ou não.

Disponibilidade: 2.0.0

Exemplo: Revestindo rasters em um quadro como bandas separadas

```
-- define our user defined function --
CREATE OR REPLACE FUNCTION raster_mapalgebra_union(
    rast1 double precision,
    rast2 double precision,
    pos integer[],
    VARIADIC userargs text[]
)
    RETURNS double precision
    AS $$
    DECLARE
    BEGIN
        CASE
            WHEN rast1 IS NOT NULL AND rast2 IS NOT NULL THEN
                RETURN ((rast1 + rast2)/2.);
            WHEN rast1 IS NULL AND rast2 IS NULL THEN
                RETURN NULL;
            WHEN rast1 IS NULL THEN
                RETURN rast2;
            ELSE
                RETURN rast1;
        END CASE;

        RETURN NULL;
    END;
    $$ LANGUAGE 'plpgsql' IMMUTABLE COST 1000;

-- prep our test table of rasters
DROP TABLE IF EXISTS map_shapes;
CREATE TABLE map_shapes(rid serial PRIMARY KEY, rast raster, bnum integer, descrip text);
INSERT INTO map_shapes(rast,bnum, descrip)
WITH mygeoms
    AS ( SELECT 2 As bnum, ST_Buffer(ST_Point(90,90),30) As geom, 'circle' As descrip
            UNION ALL
            SELECT 3 AS bnum,
                ST_Buffer(ST_GeomFromText('LINESTRING(50 50,150 150,150 50)'), 15) As geom, ↩
                    'big road' As descrip
            UNION ALL
            SELECT 1 As bnum,
                ST_Translate(ST_Buffer(ST_GeomFromText('LINESTRING(60 50,150 150,150 50)'), ↩
                    8,'join=bevel'), 10,-6) As geom, 'small road' As descrip
            ),
   -- define our canvas to be 1 to 1 pixel to geometry
   canvas
    AS ( SELECT ST_AddBand(ST_MakeEmptyRaster(250,
```

```
        250,
        ST_XMin(e)::integer, ST_YMax(e)::integer, 1, -1, 0, 0 ) , '8BUI'::text,0) As rast
        FROM (SELECT ST_Extent(geom) As e,
                    Max(ST_SRID(geom)) As srid
                    from mygeoms
                    ) As foo
            )
-- return our rasters aligned with our canvas
SELECT ST_AsRaster(m.geom, canvas.rast, '8BUI', 240) As rast, bnum, descrip
                FROM mygeoms AS m CROSS JOIN canvas
UNION ALL
SELECT canvas.rast, 4, 'canvas'
FROM canvas;

-- Map algebra on single band rasters and then collect with ST_AddBand
INSERT INTO map_shapes(rast,bnum,descrip)
SELECT ST_AddBand(ST_AddBand(rasts[1], rasts[2]),rasts[3]), 4, 'map bands overlay fct union ↩
     (canvas)'
    FROM (SELECT ARRAY(SELECT ST_MapAlgebraFct(m1.rast, m2.rast,
            'raster_mapalgebra_union(double precision, double precision, integer[], text[]) ↩
                '::regprocedure, '8BUI', 'FIRST')
                FROM map_shapes As m1 CROSS JOIN map_shapes As m2
    WHERE m1.descrip = 'canvas' AND m2.descrip <> 'canvas' ORDER BY m2.bnum) As rasts) As  ↩
        foo;
```

bandas mapa cobrem (quadro) (R: rua pequena, G: círculo, B: rua grande)

Função de usuário definido que toma argumentos extras

```
CREATE OR REPLACE FUNCTION raster_mapalgebra_userargs(
    rast1 double precision,
    rast2 double precision,
    pos integer[],
    VARIADIC userargs text[]
```

```
)
    RETURNS double precision
    AS $$
    DECLARE
    BEGIN
        CASE
            WHEN rast1 IS NOT NULL AND rast2 IS NOT NULL THEN
                RETURN least(userargs[1]::integer,(rast1 + rast2)/2.);
            WHEN rast1 IS NULL AND rast2 IS NULL THEN
                RETURN userargs[2]::integer;
            WHEN rast1 IS NULL THEN
                RETURN greatest(rast2,random()*userargs[3]::integer)::integer;
            ELSE
                RETURN greatest(rast1, random()*userargs[4]::integer)::integer;
        END CASE;

        RETURN NULL;
    END;
    $$ LANGUAGE 'plpgsql' VOLATILE COST 1000;

SELECT ST_MapAlgebraFct(m1.rast, 1, m1.rast, 3,
            'raster_mapalgebra_userargs(double precision, double precision, integer[], text ↩
                []):::regprocedure,
                '8BUI', 'INTERSECT', '100','200','200','0')
                FROM map_shapes As m1
    WHERE m1.descrip = 'map bands overlay fct union (canvas)';
```

usuário definido com argumentos extras e bandas diferentes do mesmo raster

Veja também

ST_MapAlgebraExpr, ST_BandPixelType, ST_GeoReference, ST_SetValue

10.12.11 ST_MapAlgebraFctNgb

ST_MapAlgebraFctNgb — Versão 1-banda: o vizinho mais próximo no mapa algébrico usando a função de usuário definido PostgreSQL. Retorna um raster cujos valores são o resultado de uma função usuário PLPGSQL envolvendo uma vizinhança de valores da banda raster de entrada.

Synopsis

raster **ST_MapAlgebraFctNgb**(raster rast, integer band, text pixeltype, integer ngbwidth, integer ngbheight, regprocedure onerastngbuserfunc, text nodatamode, text[] VARIADIC args);

Descrição

Warning
ST_MapAlgebraFctNgb é menosprezado como do 2.1.0. Use Funções retorno de mapa algébrico embutido.

(versão raster um) Retorna um raster cujos valores são o resultado de uma função usuário PLPGSQL envolvendo uma vizinhança de valores da banda raster de entrada. A função usuário toma a vizinhança de valores de pixel como um arranjo de números, para cada pixel, retorna o resultado da função usuário, substituindo o valor do pixel, inspecionado no momento, pelo resultado da função.

rast Raster no qual a função usuário é avaliada.

banda Número de banda do raster a ser avaliado. Padrão é 1.

pixeltype O tipo de pixel resultante do raster de saída. Deve estar listado em ST_BandPixelType ou ser deixado de fora ou ser NULO. Se não passar ou não for NULO, o padrão será o tipo de pixel do `rast`. Os resultados são cortados se eles forem maiores que o permitido para o tipo de pixel.

ngbwidth A largura da vizinhança, nas células.

ngbheight A altura da vizinhança, nas células.

onerastngbuserfunc A função usuário PLPGSQL/psq para aplicar uma vizinhança de pixeis de uma única banda de um raster. O primeiro elemento é um arranjo 2-dimensional de números representando a vizinhança do pixel retangular

nodatamode Define qual valor passar para a função para uma vizinhança de pixel que é sem dados ou NULA

'ignore': quaisquer valores NODATA encontrados na vizinhança são ignorados pelo cálculo -- esta bandeira deve ser enviada para o retorno da função usuário, e ela decide como ignorar.

'NULL': quaisquer valores NODATA encontrados na vizinhança acusamos o pixel de ser NULL -- neste caso, o retorno da função usuário é pulado.

'value': quaisquer valores NODATA encontrados na vizinhança são substituídos pelo pixel referência (o no centro da vizinhança). Note que se este valor for NODATA, o comportamento é o mesmo ce 'NULL' (para a vizinhança afetada)

args Argumentos para passar dentro da função usuário.

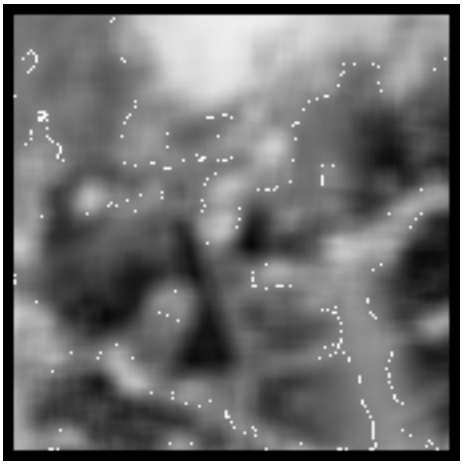
Disponibilidade: 2.0.0

Exemplos

Exemplos utilizam o raster katrina carregado como única tile descrita em http://trac.osgeo.org/gdal/wiki/frmts_wtkraster.html e preparada nos exemplos ST_Rescale

```
--
-- A simple 'callback' user function that averages up all the values in a neighborhood.
--
CREATE OR REPLACE FUNCTION rast_avg(matrix float[][], nodatamode text, variadic args text ↩
    [])
    RETURNS float AS
    $$
    DECLARE
        _matrix float[][];
        x1 integer;
        x2 integer;
        y1 integer;
        y2 integer;
        sum float;
    BEGIN
        _matrix := matrix;
        sum := 0;
        FOR x in array_lower(matrix, 1)..array_upper(matrix, 1) LOOP
            FOR y in array_lower(matrix, 2)..array_upper(matrix, 2) LOOP
                sum := sum + _matrix[x][y];
            END LOOP;
        END LOOP;
        RETURN (sum*1.0/(array_upper(matrix,1)*array_upper(matrix,2) ))::integer ;
    END;
    $$
LANGUAGE 'plpgsql' IMMUTABLE COST 1000;

-- now we apply to our raster averaging pixels within 2 pixels of each other in X and Y  ↩
    direction --
SELECT ST_MapAlgebraFctNgb(rast, 1,  '8BUI', 4,4,
        'rast_avg(float[][], text, text[])'::regprocedure, 'NULL', NULL) As nn_with_border
    FROM katrinas_rescaled
    limit 1;
```

Primeira banda do nosso raster



novo raster depois de calcular pixeis atando 4x4 pixeis de cada um

Veja também

ST_MapAlgebraFct, ST_MapAlgebraExpr, ST_Rescale

10.12.12 ST_Reclass

ST_Reclass — Cria um novo raster composto por tipos de banda reclassificados do original. A nband pode ser alterada. Se nenhuma nband for especificada, usa-se a 1. Todas as outras bandas são retornadas inalteradas. Use caso: converta uma banda 16BUI para 8BUI e então adiante para uma renderização mais simples como formatos visíveis.

Synopsis

raster **ST_Reclass**(raster rast, integer nband, text reclassexpr, text pixeltype, double precision nodataval=NULL);
raster **ST_Reclass**(raster rast, reclassarg[] VARIADIC reclassargset);
raster **ST_Reclass**(raster rast, text reclassexpr, text pixeltype);

Descrição

Cria um novo raster formado pela aplicação de uma operação algébrica válida PostgreSQL definida pela `reclassexpr` no raster de entrada (`rast`). Se nenhum `band` for especificada, usa-se a banda 1. O novo raster terá a mesma georreferência. largura e altura do raster original. As bandar não designadas voltarão inalteradas. Recorra a reclassarg para descrição de expressões de reclassificação válidas.

As bandas do novo raster terão o mesmo tipo de pixel do `pixeltype`. Se `reclassargset` passar, então, cada argumento reclassificado define o comportamento de cada banda gerada.

Disponibilidade: 2.0.0

Exemplos básicos

Cria um novo raster a partir do original onde banda 2 é convertida de 8BUI para 4BUI e todos os valores de 101-254 são definidos para valor nodata.

```
ALTER TABLE dummy_rast ADD COLUMN reclass_rast raster;
UPDATE dummy_rast SET reclass_rast = ST_Reclass(rast,2,'0-87:1-10, 88-100:11-15,  ↩
    101-254:0-0', '4BUI',0) WHERE rid = 2;

SELECT i as col, j as row, ST_Value(rast,2,i,j) As origval,
    ST_Value(reclass_rast, 2, i, j) As reclassval,
    ST_Value(reclass_rast, 2, i, j, false) As reclassval_include_nodata
FROM dummy_rast CROSS JOIN generate_series(1, 3) AS i CROSS JOIN generate_series(1,3) AS j
WHERE rid = 2;

 col | row | origval | reclassval | reclassval_include_nodata
-----+-----+---------+------------+--------------------------
   1 |   1 |      78 |          9 |                         9
   2 |   1 |      98 |         14 |                        14
   3 |   1 |     122 |            |                         0
   1 |   2 |      96 |         14 |                        14
   2 |   2 |     118 |            |                         0
   3 |   2 |     180 |            |                         0
   1 |   3 |      99 |         15 |                        15
   2 |   3 |     112 |            |                         0
   3 |   3 |     169 |            |                         0
```

Exemplo: Uso avançado de múltiplos reclassargs

Cria um novo raster do original onde banda 1,2,3 é convertida para 1BB, 4BUI, 4BUI respectivamente e reclassificada. Note que isto usa o argumento variado `reclassarg` que pode pegar como entrada e número indefinido de reclssargs (teoricamente quantas bandas tiver)

```
UPDATE dummy_rast SET reclass_rast =
    ST_Reclass(rast,
        ROW(2,'0-87]:1-10, (87-100]:11-15, (101-254]:0-0', '4BUI',NULL)::reclassarg,
        ROW(1,'0-253]:1, 254:0', '1BB', NULL)::reclassarg,
        ROW(3,'0-70]:1, (70-86:2, [86-150):3, [150-255:4', '4BUI', NULL)::reclassarg
        ) WHERE rid = 2;

SELECT i as col, j as row,ST_Value(rast,1,i,j) As ov1,  ST_Value(reclass_rast, 1, i, j) As ↩
    rv1,
    ST_Value(rast,2,i,j) As ov2, ST_Value(reclass_rast, 2, i, j) As rv2,
    ST_Value(rast,3,i,j) As ov3, ST_Value(reclass_rast, 3, i, j) As rv3
FROM dummy_rast CROSS JOIN generate_series(1, 3) AS i CROSS JOIN generate_series(1,3) AS j
WHERE rid = 2;

col | row | ov1 | rv1 | ov2 | rv2 | ov3 | rv3
----+-----+-----+-----+-----+-----+-----+-----
  1 |   1 | 253 |   1 |  78 |   9 |  70 |   1
  2 |   1 | 254 |   0 |  98 |  14 |  86 |   3
  3 |   1 | 253 |   1 | 122 |   0 | 100 |   3
  1 |   2 | 253 |   1 |  96 |  14 |  80 |   2
  2 |   2 | 254 |   0 | 118 |   0 | 108 |   3
  3 |   2 | 254 |   0 | 180 |   0 | 162 |   4
  1 |   3 | 250 |   1 |  99 |  15 |  90 |   3
  2 |   3 | 254 |   0 | 112 |   0 | 108 |   3
  3 |   3 | 254 |   0 | 169 |   0 | 175 |   4
```

Exemplo: Mapeamento avançado de uma única banda raster 32BF para multiplicar bandas visíveis

Cria uma nova banda 3 (8BUI,8BUI,8BUI raster visível) a partir de um raster que tem apena uma banda 32bf

```
ALTER TABLE wind ADD COLUMN rast_view raster;
UPDATE wind
    set rast_view = ST_AddBand( NULL,
        ARRAY[
    ST_Reclass(rast, 1,'0.1-10]:1-10,9-10]:11,(11-33:0'::text, '8BUI'::text,0),
    ST_Reclass(rast,1, '11-33):0-255,[0-32:0,(34-1000:0'::text, '8BUI'::text,0),
    ST_Reclass(rast,1,'0-32]:0,(32-100:100-255'::text, '8BUI'::text,0)
    ]
    );
```

Veja também

ST_AddBand, ST_Band, ST_BandPixelType, ST_MakeEmptyRaster, reclassarg, ST_Value

10.12.13 ST_Union

ST_Union — Retorna a união de um conjunto de tiles raster em um único raster composto de 1 ou mais bandas.

Synopsis

raster **ST_Union**(setof raster rast);
raster **ST_Union**(setof raster rast, unionarg[] unionargset);
raster **ST_Union**(setof raster rast, integer nband);
raster **ST_Union**(setof raster rast, text uniontype);
raster **ST_Union**(setof raster rast, integer nband, text uniontype);

Descrição

Retorna a união de um conjunto de tiles raster em um único raster composto de pelo menos uma banda. A extensão resultante do raster é a extensão do conjunto todo. No caso da interseção, o valor resultante é definido pelo `uniontype` que é um dos seguintes: LAST (default), FIRST, MIN, MAX, COUNT, SUM, MEAN, RANGE.

Note
In order for rasters to be unioned, they must all have the same alignment. Use ST_SameAlignment and ST_NotSameAlignmentReason for more details and help. One way to fix alignment issues is to use ST_Resample and use the same reference raster for alignment.

Disponibilidade: 2.0.0

Melhorias: 2.1.0 Velocidade aprimorada (fully C-Based)

Disponibilidade: 2.1.0 variante ST_Union(rast, unionarg) foi introduzida.

Melhorias: 2.1.0 uniões ST_Union(rast) (variante 1) todas as bandas de todos os rasters de entrada. As versões anteriores do PostGIS assumiam a primeira banda.

Melhorias: 2.1.0 ST_Union(rast, uniontype) (variante 4) uniões de todas as bandas de todos os rasters de entrada.

Exemplos: Reconstitui uma única tile banda raster em pedaços

```
-- this creates a single band from first band of raster tiles
-- that form the original file system tile
SELECT filename, ST_Union(rast,1) As file_rast
FROM sometable WHERE filename IN('dem01', 'dem02') GROUP BY filename;
```

Exemplos: Retorna uma multi banda raster que é a união de tiles intersectando geometrias

```
-- this creates a multi band raster collecting all the tiles that intersect a line
-- Note: In 2.0, this would have just returned a single band raster
-- , new union works on all bands by default
-- this is equivalent to unionarg: ARRAY[ROW(1, 'LAST'), ROW(2, 'LAST'), ROW(3, 'LAST')]:: ↩
    unionarg[]
SELECT ST_Union(rast)
FROM aerials.boston
WHERE ST_Intersects(rast,  ST_GeomFromText('LINESTRING(230486 887771, 230500 88772)',26986) ↩
    );
```

Exemplos: Retorna uma multi banda raster que é a união de tiles intersectando geometrias

Aqui, usamos a sintaxe mais longa se só queremos uma subset de bandas ou queremos alterar a ordem das bandas

```
-- this creates a multi band raster collecting all the tiles that intersect a line
SELECT ST_Union(rast,ARRAY[ROW(2, 'LAST'), ROW(1, 'LAST'), ROW(3, 'LAST')]::unionarg[])
FROM aerials.boston
WHERE ST_Intersects(rast,  ST_GeomFromText('LINESTRING(230486 887771, 230500 88772)',26986) ↩
    );
```

Veja também

unionarg, ST_Envelope, ST_ConvexHull, ST_Clip, ST_Union

10.13 Funções retorno de mapa algébrico embutido

10.13.1 ST_Distinct4ma

ST_Distinct4ma — Função de processamento raster que calcula o resumo de valores únicos de pixel em uma vizinhança.

Synopsis

float8 **ST_Distinct4ma**(float8[][] matrix, text nodatamode, text[] VARIADIC args);
double precision **ST_Distinct4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

Descrição

Calcula o número de valores únicos de pixel em uma vizinhança.

Note
A variante 1 é uma função retorno especializada em uso como um parâmetro de retorno para ST_MapAlgebraFctNgb.

Note
A variante 2 é uma função retorno especializada em uso como um parâmetro de retorno para Funções retorno de mapa algébrico embutido.

Warning
Uso da variante 1 é desencorajado desde que ST_MapAlgebraFctNgb foi menosprezada como de 2.1.0.

Disponibilidade: 2.0.0

Melhorias: 2.1.0 Adição da variante 2

Exemplos

```
SELECT
    rid,
    st_value(
        st_mapalgebrafctngb(rast, 1, NULL, 1, 1, 'st_distinct4ma(float[][],text,text[])':: ↩
            regprocedure, 'ignore', NULL), 2, 2
    )
FROM dummy_rast
WHERE rid = 2;
 rid | st_value
-----+----------
   2 |        3
(1 row)
```

Veja também

ST_MapAlgebraFctNgb, Funções retorno de mapa algébrico embutido, ST_Min4ma, ST_Max4ma, ST_Sum4ma, ST_Mean4ma, ST_Distinct4ma, ST_StdDev4ma

10.13.2 ST_InvDistWeight4ma

ST_InvDistWeight4ma — Função de processamento raster que interpola um valor de pixel de uma vizinhança.

Synopsis

double precision **ST_InvDistWeight4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

Descrição

Calcula um valor interpolado para um pixel usando o método do inverso da potência das distâncias.

Existem dois parâmetros opcionais que podem ser passados pelos `userargs`. O primeiro parâmetro é o fator de força (variável k na equação abaixo) entre 0 e 1 usado na equação do inverso da potência das distâncias. Se não especificado, usa-se 1. O segundo parâmetro é a porcentagem aplicada somente quando o valor do pixel de interesse estiver incluso no valor da vizinhança. Se não especificado e o pixel de interesse possuir um valor, o valor é retornado.

A equação do inverso da potência das distâncias é:

$$\hat{z}(x_o) = \frac{\sum_{j=1}^{m} z(x_j) d_{ij}^{-k}}{\sum_{j=1}^{m} d_{ij}^{-k}}$$

k = fator força, um número real entre 0 e 1

Note
Esta função é uma função retorno especializada em uso como parâmetro de retorno para Funções retorno de mapa algébrico embutido.

Disponibilidade: 2.1.0

Exemplos

```
-- NEEDS EXAMPLE
```

Veja também

Funções retorno de mapa algébrico embutido, ST_MinDist4ma

10.13.3 ST_Max4ma

ST_Max4ma — Função de processamento raster que calcula o valor máximo de pixel em uma vizinhança.

Synopsis

float8 **ST_Max4ma**(float8[][] matrix, text nodatamode, text[] VARIADIC args);
double precision **ST_Max4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

Descrição

Calcula o valor de pixel máximo em uma vizinhança de pixeis.

Para Variante 2, um valor de substituição para pixeis NODATA podem ser especificados passando aquele valor para userargs.

Note
A variante 1 é uma função retorno especializada em uso como um parâmetro de retorno para ST_MapAlgebraFctNgb.

Note
A variante 2 é uma função retorno especializada em uso como um parâmetro de retorno para Funções retorno de mapa algébrico embutido.

Warning
Uso da variante 1 é desencorajado desde que ST_MapAlgebraFctNgb foi menosprezada como de 2.1.0.

Disponibilidade: 2.0.0

Melhorias: 2.1.0 Adição da variante 2

Exemplos

```
SELECT
    rid,
    st_value(
        st_mapalgebrafctngb(rast, 1, NULL, 1, 1, 'st_max4ma(float[][],text,text[])':: ↩
            regprocedure, 'ignore', NULL), 2, 2
    )
FROM dummy_rast
```

```
WHERE rid = 2;
 rid | st_value
-----+----------
   2 |      254
(1 row)
```

Veja também

ST_MapAlgebraFctNgb, Funções retorno de mapa algébrico embutido, ST_Min4ma, ST_Sum4ma, ST_Mean4ma, ST_Range4ma, ST_Distinct4ma, ST_StdDev4ma

10.13.4 ST_Mean4ma

ST_Mean4ma — Função de processamento raster que calcula o menor valor de pixel em uma vizinhança.

Synopsis

float8 **ST_Mean4ma**(float8[][] matrix, text nodatamode, text[] VARIADIC args);
double precision **ST_Mean4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

Descrição

Calcula o menor valor de pixel em uma vizinhança de pixeis.

Para Variante 2, um valor de substituição para pixeis NODATA podem ser especificados passando aquele valor para userargs.

Note
A variante 1 é uma função retorno especializada em uso como um parâmetro de retorno para ST_MapAlgebraFctNgb.

Note
A variante 2 é uma função retorno especializada em uso como um parâmetro de retorno para Funções retorno de mapa algébrico embutido.

Warning
Uso da variante 1 é desencorajado desde que ST_MapAlgebraFctNgb foi menosprezada como de 2.1.0.

Disponibilidade: 2.0.0

Melhorias: 2.1.0 Adição da variante 2

Exemplos: Variante 1

```
SELECT
    rid,
    st_value(
        st_mapalgebrafctngb(rast, 1, '32BF', 1, 1, 'st_mean4ma(float[][],text,text[])':: ↩
            regprocedure, 'ignore', NULL), 2, 2
    )
FROM dummy_rast
WHERE rid = 2;
 rid |     st_value
-----+------------------
   2 | 253.222229003906
(1 row)
```

Exemplos: Variant 2

```
SELECT
    rid,
    st_value(
               ST_MapAlgebra(rast, 1, 'st_mean4ma(double precision[][][], integer[][], text ↩
                   [])'::regprocedure,'32BF', 'FIRST', NULL, 1, 1)
        ,  2, 2)
  FROM dummy_rast
   WHERE rid = 2;
 rid |     st_value
-----+------------------
   2 | 253.222229003906
(1 row)
```

Veja também

ST_MapAlgebraFctNgb, Funções retorno de mapa algébrico embutido, ST_Min4ma, ST_Max4ma, ST_Sum4ma, ST_Range4ma, ST_StdDev4ma

10.13.5 ST_Min4ma

ST_Min4ma — Função de processamento raster que calcula o valor mínimo de pixel em uma vizinhança.

Synopsis

float8 **ST_Min4ma**(float8[][] matrix, text nodatamode, text[] VARIADIC args);
double precision **ST_Min4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

Descrição

Calcula o valor de pixel mínimo em uma vizinhança de pixeis.

Para Variante 2, um valor de substituição para pixeis NODATA podem ser especificados passando aquele valor para userargs.

Note
A variante 1 é uma função retorno especializada em uso como um parâmetro de retorno para ST_MapAlgebraFctNgb.

Note
A variante 2 é uma função retorno especializada em uso como um parâmetro de retorno para Funções retorno de mapa algébrico embutido.

Warning
Uso da variante 1 é desencorajado desde que ST_MapAlgebraFctNgb foi menosprezada como de 2.1.0.

Disponibilidade: 2.0.0

Melhorias: 2.1.0 Adição da variante 2

Exemplos

```
SELECT
    rid,
    st_value(
        st_mapalgebrafctngb(rast, 1, NULL, 1, 1, 'st_min4ma(float[][],text,text[])':: ↩
            regprocedure, 'ignore', NULL), 2, 2
    )
FROM dummy_rast
WHERE rid = 2;
 rid | st_value
-----+----------
   2 |      250
(1 row)
```

Veja também

ST_MapAlgebraFctNgb, Funções retorno de mapa algébrico embutido, ST_Max4ma, ST_Sum4ma, ST_Mean4ma, ST_Range4ma, ST_Distinct4ma, ST_StdDev4ma

10.13.6 ST_MinDist4ma

ST_MinDist4ma — Função de processamento raster que retorna a distância mínima (em números de pixeis) entre o pixel de interesse e um pixel vizinho de interesse com valor.

Synopsis

double precision **ST_MinDist4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

Descrição

Retorna a menor função (em números de pixeis) entre o pixel de interesse e o pixel mais próximo com valor na vizinhança.

Note
A intenção desta função é fornecer um ponto de dados informativos que ajude inferir a utilidade do valor interpolado do pixel de interesse da ST_InvDistWeight4ma. Esta função é particularmente útil quando a vizinhança é esparsamente populada.

Note
Esta função é uma função retorno especializada em uso como parâmetro de retorno para Funções retorno de mapa algébrico embutido.

Disponibilidade: 2.1.0

Exemplos

```
-- NEEDS EXAMPLE
```

Veja também

Funções retorno de mapa algébrico embutido, ST_InvDistWeight4ma

10.13.7 ST_Range4ma

ST_Range4ma — Função de processamento raster que calcula a varição de valores de pixel em uma vizinhança.

Synopsis

float8 **ST_Range4ma**(float8[][] matrix, text nodatamode, text[] VARIADIC args);
double precision **ST_Range4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

Descrição

Calcula a variação de valores de pixel em uma vizinhança de pixeis.

Para Variante 2, um valor de substituição para pixeis NODATA podem ser especificados passando aquele valor para userargs.

Note
A variante 1 é uma função retorno especializada em uso como um parâmetro de retorno para ST_MapAlgebraFctNgb.

Note
A variante 2 é uma função retorno especializada em uso como um parâmetro de retorno para Funções retorno de mapa algébrico embutido.

Warning
Uso da variante 1 é desencorajado desde que ST_MapAlgebraFctNgb foi menosprezada como de 2.1.0.

Disponibilidade: 2.0.0

Melhorias: 2.1.0 Adição da variante 2

Exemplos

```
SELECT
    rid,
    st_value(
        st_mapalgebrafctngb(rast, 1, NULL, 1, 1, 'st_range4ma(float[][],text,text[])':: ↩
            regprocedure, 'ignore', NULL), 2, 2
    )
FROM dummy_rast
WHERE rid = 2;
 rid | st_value
-----+----------
   2 |        4
(1 row)
```

Veja também

ST_MapAlgebraFctNgb, Funções retorno de mapa algébrico embutido, ST_Min4ma, ST_Max4ma, ST_Sum4ma, ST_Mean4ma, ST_Distinct4ma, ST_StdDev4ma

10.13.8 ST_StdDev4ma

ST_StdDev4ma — Função de processamento raster que calcula o padrão de divergência de valores de pixel em uma vizinhança.

Synopsis

float8 **ST_StdDev4ma**(float8[][] matrix, text nodatamode, text[] VARIADIC args);
double precision **ST_StdDev4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

Descrição

Calcula o padrão de divergência de valores de pixel em uma vizinhança de pixeis.

Note
A variante 1 é uma função retorno especializada em uso como um parâmetro de retorno para ST_MapAlgebraFctNgb.

Note
A variante 2 é uma função retorno especializada em uso como um parâmetro de retorno para Funções retorno de mapa algébrico embutido.

Warning
Uso da variante 1 é desencorajado desde que ST_MapAlgebraFctNgb foi menosprezada como de 2.1.0.

Disponibilidade: 2.0.0

Melhorias: 2.1.0 Adição da variante 2

Exemplos

```
SELECT
    rid,
    st_value(
        st_mapalgebrafctngb(rast, 1, '32BF', 1, 1, 'st_stddev4ma(float[][],text,text[])':: ↩
            regprocedure, 'ignore', NULL), 2, 2
    )
FROM dummy_rast
WHERE rid = 2;
 rid |     st_value
-----+------------------
   2 | 1.30170822143555
(1 row)
```

Veja também

ST_MapAlgebraFctNgb, Funções retorno de mapa algébrico embutido, ST_Min4ma, ST_Max4ma, ST_Sum4ma, ST_Mean4ma, ST_Distinct4ma, ST_StdDev4ma

10.13.9 ST_Sum4ma

ST_Sum4ma — Função de processamento raster que calcula o resumo de todos os valores de pixel em uma vizinhança.

Synopsis

float8 **ST_Sum4ma**(float8[][] matrix, text nodatamode, text[] VARIADIC args);
double precision **ST_Sum4ma**(double precision[][][] value, integer[][] pos, text[] VARIADIC userargs);

Descrição

Calcula o resumo de todos os valores de pixel em uma vizinhança de pixeis.

Para Variante 2, um valor de substituição para pixeis NODATA podem ser especificados passando aquele valor para userargs.

Note
A variante 1 é uma função retorno especializada em uso como um parâmetro de retorno para ST_MapAlgebraFctNgb.

Note
A variante 2 é uma função retorno especializada em uso como um parâmetro de retorno para Funções retorno de mapa algébrico embutido.

Warning
Uso da variante 1 é desencorajado desde que ST_MapAlgebraFctNgb foi menosprezada como de 2.1.0.

Disponibilidade: 2.0.0

Melhorias: 2.1.0 Adição da variante 2

Exemplos

```
SELECT
    rid,
    st_value(
        st_mapalgebrafctngb(rast, 1, '32BF', 1, 1, 'st_sum4ma(float[][],text,text[])':: ↩
            regprocedure, 'ignore', NULL), 2, 2
    )
FROM dummy_rast
WHERE rid = 2;
 rid | st_value
-----+----------
   2 |     2279
(1 row)
```

Veja também

ST_MapAlgebraFctNgb, Funções retorno de mapa algébrico embutido, ST_Min4ma, ST_Max4ma, ST_Mean4ma, ST_Range4ma, ST_Distinct4ma, ST_StdDev4ma

10.14 Processamento Raster

10.14.1 ST_Aspect

ST_Aspect — Retorna o aspecto (em graus) de uma banda raster de elevação. Útil para analisar terrenos.

Synopsis

raster **ST_Aspect**(raster rast, integer band=1, text pixeltype=32BF, text units=DEGREES, boolean interpolate_nodata=FALSE);
raster **ST_Aspect**(raster rast, integer band, raster customextent, text pixeltype=32BF, text units=DEGREES, boolean interpolate_nodata=FALSE);

Descrição

Retorna o aspecto (em graus) de uma banda raster de elevação. Utiliza mapa algébrico e aplica o aspecto de equação para pixeis vizinhos.

`units` indica as unidade do aspecto. Possíveis valores são: RADIANOS, GRAUS (padrão).

Quando `units` = RADIANOS, valoers são entre 0 e 2 * pi radianos medidos sentido horário a partir do Norte.

Quando `units` = GRAUS, valores são entre 0 e 360 graus medidos a partir do Norte.

Se o declive de pixel for zero, o aspecto do pixel é -1.

Note
Para maiores informações sobre declive, aspecto e sombreado, por favor recorra a ESRI - How hillshade works e ERDAS Field Guide - Aspect Images.

Disponibilidade: 2.0.0

melhorias: 2.1.0 Usa ST_MapAlgebra() e foi adicionado uma função parâmetro opcional `interpolate_nodata`

Alterações: 2.1.0 Nas versões anteriores, os valores retornados eram em radianos. Agora, eles retornam em graus

Exemplos: Variante 1

```
WITH foo AS (
    SELECT ST_SetValues(
        ST_AddBand(ST_MakeEmptyRaster(5, 5, 0, 0, 1, -1, 0, 0, 0), 1, '32BF', 0, -9999),
        1, 1, 1, ARRAY[
            [1, 1, 1, 1, 1],
            [1, 2, 2, 2, 1],
            [1, 2, 3, 2, 1],
            [1, 2, 2, 2, 1],
            [1, 1, 1, 1, 1]
        ]::double precision[][]
    ) AS rast
)
SELECT
    ST_DumpValues(ST_Aspect(rast, 1, '32BF'))
FROM foo




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

-----------------------------------
 (1,"{{315,341.565063476562,0,18.4349479675293,45},{288.434936523438,315,0,45,71.5650482177734},{270

2227,180,161.565048217773,135}}")
(1 row)
```

Exemplos: Variant 2

Exemplo completo de tiles de uma cobertura. Esta consulta funciona apenas com PostgreSQL 9.1 ou superior.

```
WITH foo AS (
    SELECT ST_Tile(
        ST_SetValues(
            ST_AddBand(
                ST_MakeEmptyRaster(6, 6, 0, 0, 1, -1, 0, 0, 0),
                1, '32BF', 0, -9999
            ),
            1, 1, 1, ARRAY[
                [1, 1, 1, 1, 1, 1],
                [1, 1, 1, 1, 2, 1],
                [1, 2, 2, 3, 3, 1],
                [1, 1, 3, 2, 1, 1],
                [1, 2, 2, 1, 2, 1],
                [1, 1, 1, 1, 1, 1]
            ]::double precision[]
        ),
        2, 2
    ) AS rast
)
SELECT
    t1.rast,
    ST_Aspect(ST_Union(t2.rast), 1, t1.rast)
FROM foo t1
CROSS JOIN foo t2
WHERE ST_Intersects(t1.rast, t2.rast)
GROUP BY t1.rast;
```

Veja também

Funções retorno de mapa algébrico embutido, ST_TRI, ST_TPI, ST_Roughness, ST_HillShade, ST_Slope

10.14.2 ST_HillShade

ST_HillShade — Retorna a iluminação hipotética de uma banda raster de elevação usando as entradas de azimute, altitude, claridade e escala fornecidas.

Synopsis

raster **ST_HillShade**(raster rast, integer band=1, text pixeltype=32BF, double precision azimuth=315, double precision altitude=45, double precision max_bright=255, double precision scale=1.0, boolean interpolate_nodata=FALSE);
raster **ST_HillShade**(raster rast, integer band, raster customextent, text pixeltype=32BF, double precision azimuth=315, double precision altitude=45, double precision max_bright=255, double precision scale=1.0, boolean interpolate_nodata=FALSE);

Descrição

Retorna a iluminação hipotética de uma banda raster de elevação usando as entradas de azimute, altitude, claridade e escala fornecidas. Utiliza mapa algébrico e aplica a equação sombreada nos pixeis vizinhos. Os valores de pixel retornados estão entre 0 e 255.

`azimuth` é um valor entre 0 e 360 graus medidos no sentido horário a partir do Norte.

`altitude` é um valor entre 0 e 90 graus onde 0 grau está no horizonte e 90 graus estão diretamente em cima.

`max_bright` é um valor entre 0 e 255 com 0 sendo nenhuma claridade e 255 sendo a claridade máxima.

`scale` is the ratio of vertical units to horizontal. For Feet:LatLon use scale=370400, for Meters:LatLon use scale=111120.

Se `interpolate_nodata` for VERDADE, valores para pixeis NODATA do raster de entrada serão interpolados usando ST_InvDistWeight4ma antes de calcular a iluminação sombreada.

Note
para maiores informações sobre sombreamento, por favor recorra a How hillshade works.

Disponibilidade: 2.0.0

melhorias: 2.1.0 Usa ST_MapAlgebra() e foi adicionado uma função parâmetro opcional `interpolate_nodata`

Alterações: 2.1.0 Nas versões anteriores, o azimute e a altitude eram expressados em radianos. Agora, são representados em graus

Exemplos: Variante 1

```
WITH foo AS (
    SELECT ST_SetValues(
        ST_AddBand(ST_MakeEmptyRaster(5, 5, 0, 0, 1, -1, 0, 0, 0), 1, '32BF', 0, -9999),
        1, 1, 1, ARRAY[
            [1, 1, 1, 1, 1],
            [1, 2, 2, 2, 1],
            [1, 2, 3, 2, 1],
            [1, 2, 2, 2, 1],
            [1, 1, 1, 1, 1]
        ]::double precision[][]
    ) AS rast
```

```
)
SELECT
    ST_DumpValues(ST_Hillshade(rast, 1, '32BF'))
FROM foo




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


-----------------------------------------------------------------------
 (1,"{{NULL,NULL,NULL,NULL,NULL},{NULL,251.32763671875,220.749786376953,147.224319458008, ↩
     NULL},{NULL,220.749786376953,180.312225341797,67.7497863769531,NULL},{NULL ↩
     ,147.224319458008
,67.7497863769531,43.1210060119629,NULL},{NULL,NULL,NULL,NULL,NULL}}")
(1 row)
```

Exemplos: Variant 2

Exemplo completo de tiles de uma cobertura. Esta consulta funciona apenas com PostgreSQL 9.1 ou superior.

```
WITH foo AS (
    SELECT ST_Tile(
        ST_SetValues(
            ST_AddBand(
                ST_MakeEmptyRaster(6, 6, 0, 0, 1, -1, 0, 0, 0),
                1, '32BF', 0, -9999
            ),
            1, 1, 1, ARRAY[
                [1, 1, 1, 1, 1, 1],
                [1, 1, 1, 1, 2, 1],
                [1, 2, 2, 3, 3, 1],
                [1, 1, 3, 2, 1, 1],
                [1, 2, 2, 1, 2, 1],
                [1, 1, 1, 1, 1, 1]
            ]::double precision[]
        ),
        2, 2
    ) AS rast
)
SELECT
    t1.rast,
    ST_Hillshade(ST_Union(t2.rast), 1, t1.rast)
FROM foo t1
CROSS JOIN foo t2
WHERE ST_Intersects(t1.rast, t2.rast)
GROUP BY t1.rast;
```

Veja também

Funções retorno de mapa algébrico embutido, ST_TRI, ST_TPI, ST_Roughness, ST_Aspect, ST_Slope

10.14.3 ST_Roughness

ST_Roughness — Retorna um raster com a "robustez" calculada de um DEM.

Synopsis

raster **ST_Roughness**(raster rast, integer nband, raster customextent, text pixeltype="32BF" , boolean interpolate_nodata=FALSE);

Descrição

Calcula a "robustez" de um DEM, subtraindo o máximo do mínimo de uma dada área.

Disponibilidade: 2.1.0

Exemplos

```
-- needs examples
```

Veja também

Funções retorno de mapa algébrico embutido, ST_TRI, ST_TPI, ST_Slope, ST_HillShade, ST_Aspect

10.14.4 ST_Slope

ST_Slope — Retorna o declive (em graus) de uma banda raster de elevação. Útil para analisar terrenos.

Synopsis

raster **ST_Slope**(raster rast, integer nband=1, text pixeltype=32BF, text units=DEGREES, double precision scale=1.0, boolean interpolate_nodata=FALSE);
raster **ST_Slope**(raster rast, integer nband, raster customextent, text pixeltype=32BF, text units=DEGREES, double precision scale=1.0, boolean interpolate_nodata=FALSE);

Descrição

Retorna o declive (em graus) de uma banda raster de elevação. Utiliza mapa algébrico e aplica a equação de declive nos pixeis vizinhos.

`units` indica as unidades do declive. Possíveis valores são: RADIANOS, GRAUS (padrão), PORCENTAGEM.

`scale` is the ratio of vertical units to horizontal. For Feet:LatLon use scale=370400, for Meters:LatLon use scale=111120.

Se `interpolate_nodata` for VERDADE, valores para pixeis NODATA do raster de entrada serão interpolados usando ST_InvDistWeight4ma antes de calcular a superfície inclinada.

Note
Para maiores informações sobre declive, aspecto e sombreado, por favor recorra a ESRI - How hillshade works and ERDAS Field Guide - Slope Images.

Disponibilidade: 2.0.0

Melhorias: 2.1.0 Usa ST_MapAlgebra() e foi adicionado a função parâmetros opcionais `units`, `scale`, `interpolate_nodata`

Alterações: 2.1.0 Nas versões anteriores, os valores retornados eram em radianos. Agora, eles retornam em graus

Exemplos: Variante 1

```
WITH foo AS (
    SELECT ST_SetValues(
        ST_AddBand(ST_MakeEmptyRaster(5, 5, 0, 0, 1, -1, 0, 0, 0), 1, '32BF', 0, -9999),
        1, 1, 1, ARRAY[
            [1, 1, 1, 1, 1],
            [1, 2, 2, 2, 1],
            [1, 2, 3, 2, 1],
            [1, 2, 2, 2, 1],
            [1, 1, 1, 1, 1]
        ]::double precision[][]
    ) AS rast
)
SELECT
    ST_DumpValues(ST_Slope(rast, 1, '32BF'))
FROM foo

                            st_dumpvalues

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

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

-------------------------------------------------------------------
 (1,"{{10.0249881744385,21.5681285858154,26.5650520324707,21.5681285858154,10.0249881744385},{21.568

{26.5650520324707,36.8698959350586,0,36.8698959350586,26.5650520324707},{21.5681285858154,35.2643890

5681285858154,26.5650520324707,21.5681285858154,10.0249881744385}}")
(1 row)
```

Exemplos: Variant 2

Exemplo completo de tiles de uma cobertura. Esta consulta funciona apenas com PostgreSQL 9.1 ou superior.

```
WITH foo AS (
    SELECT ST_Tile(
        ST_SetValues(
            ST_AddBand(
                ST_MakeEmptyRaster(6, 6, 0, 0, 1, -1, 0, 0, 0),
                1, '32BF', 0, -9999
            ),
            1, 1, 1, ARRAY[
                [1, 1, 1, 1, 1, 1],
                [1, 1, 1, 1, 2, 1],
                [1, 2, 2, 3, 3, 1],
                [1, 1, 3, 2, 1, 1],
                [1, 2, 2, 1, 2, 1],
                [1, 1, 1, 1, 1, 1]
            ]::double precision[]
        ),
        2, 2
    ) AS rast
)
SELECT
    t1.rast,
    ST_Slope(ST_Union(t2.rast), 1, t1.rast)
FROM foo t1
CROSS JOIN foo t2
```

```
WHERE ST_Intersects(t1.rast, t2.rast)
GROUP BY t1.rast;
```

Veja também

Funções retorno de mapa algébrico embutido, ST_TRI, ST_TPI, ST_Roughness, ST_HillShade, ST_Aspect

10.14.5 ST_TPI

ST_TPI — Retorna um raster com o índice de posição topográfico calculado.

Synopsis

raster **ST_TPI**(raster rast, integer nband, raster customextent, text pixeltype="32BF" , boolean interpolate_nodata=FALSE);

Descrição

Calculates the Topographic Position Index, which is defined as the focal mean with radius of one minus the center cell.

Note
Esta função suporta apenas o raio mínimo central.

Disponibilidade: 2.1.0

Exemplos

```
-- needs examples
```

Veja também

Funções retorno de mapa algébrico embutido, ST_TRI, ST_Roughness, ST_Slope, ST_HillShade, ST_Aspect

10.14.6 ST_TRI

ST_TRI — Retorna um raster com o índice de aspereza do terreno calculado.

Synopsis

raster **ST_TRI**(raster rast, integer nband, raster customextent, text pixeltype="32BF" , boolean interpolate_nodata=FALSE);

Descrição

O índice de aspereza do terreno é calculado pela comparação de um pixel central com seus vizinhos, pegando os valores absolutos das diferenças, e calculando o resultado.

Note
Esta função suporta apenas o raio mínimo central.

Disponibilidade: 2.1.0

Exemplos

```
-- needs examples
```

Veja também

Funções retorno de mapa algébrico embutido, ST_Roughness, ST_TPI, ST_Slope, ST_HillShade, ST_Aspect

10.15 Raster para Geometria

10.15.1 Caixa3D

Caixa3D — Retorna a representação da caixa 3d da caixa encerrada do raster.

Synopsis

box3d **Box3D**(raster rast);

Descrição

Retorna a caixa representando a extensão do raster.

O polígono é definido pelos pontos de canto da caixa delimitadora ((MINX, MINY), (MAXX, MAXY))

Alterações: 2.0.0 Nas versões pre-2.0, costumava existir uma caixa2d em vez de uma caixa3d. Já que a caixa2d é um tipo inferior, foi alterado para caixa3d.

Exemplos

```
SELECT
    rid,
    Box3D(rast) AS rastbox
FROM dummy_rast;

rid |        rastbox
----+-------------------------------------------------
1   | BOX3D(0.5 0.5 0,20.5 60.5 0)
2   | BOX3D(3427927.75 5793243.5 0,3427928 5793244 0)
```

Veja também

ST_Envelope

10.15.2 ST_ConvexHull

ST_ConvexHull — Retorna o casco convexo da geometria do raster incluindo valores iguais ao BandNoDataValue. Para rasters com formas normais e não desviadas, o resultado é o mesmo que ST_Envelope, então só é útil para rasters com formas irregulares ou desviados.

Synopsis

geometry **ST_ConvexHull**(raster rast);

Descrição

Retorna o casco convexo da geometria do raster incluindo valores iguais ao NoDataBandValue pixeis banda. Para rasters com formas normais e não desviadas, o resultado é o mesmo que ST_Envelope, então só é útil para rasters com formas irregulares ou desviados.

Note
ST_Envelope derruba as coordenadas e por isso adiciona um pequeno buffer em torno do raster, então a resposta é um pouco diferente da ST_ConvexHull que não derruba.

Exemplos

Recorra a PostGIS Raster Specification para um diagrama.

```
-- Note envelope and convexhull are more or less the same
SELECT ST_AsText(ST_ConvexHull(rast)) As convhull,
    ST_AsText(ST_Envelope(rast)) As env
FROM dummy_rast WHERE rid=1;

                        convhull                         |                env
---------------------------------------------------------+----------------------------------- ↩

 POLYGON((0.5 0.5,20.5 0.5,20.5 60.5,0.5 60.5,0.5 0.5)) | POLYGON((0 0,20 0,20 60,0 60,0 0) ↩
     )
```

```
-- now we skew the raster
-- note how the convex hull and envelope are now different
SELECT ST_AsText(ST_ConvexHull(rast)) As convhull,
    ST_AsText(ST_Envelope(rast)) As env
FROM (SELECT ST_SetRotation(rast, 0.1, 0.1) As rast
    FROM dummy_rast WHERE rid=1) As foo;

                        convhull                         |                env
---------------------------------------------------------+----------------------------------- ↩

 POLYGON((0.5 0.5,20.5 1.5,22.5 61.5,2.5 60.5,0.5 0.5)) | POLYGON((0 0,22 0,22 61,0 61,0 0) ↩
     )
```

Veja também

ST_Envelope, ST_MinConvexHull, ST_ConvexHull, ST_AsText

10.15.3 ST_DumpAsPolygons

ST_DumpAsPolygons — Retorna um conjunto de linhas geomval (geom,val), de uma dada banda raster. Se nenhum número de banda for especificado, o número de banda torna-se 1.

Synopsis

setof geomval **ST_DumpAsPolygons**(raster rast, integer band_num=1, boolean exclude_nodata_value=TRUE);

Descrição

Esta é uma função retorno (SRF). Ela retorna um conjunto de linhas geomval, formadas por uma geometria (geom) e uma banda pixel valor (val). Cada polígono é a união de todos os pixeis para aquela banda que tem o mesmo valor de pixel indicado pelo val.

ST_DumpAsPolygon é útil para poligonizar rasters. É o reverso de um GRUPO POR onde cria novas filas. Por exemplo, pode ser usada para expandir um único raster em POLÍGONOS/MULTIPOLÍGONOS.

Changed 3.3.0, validation and fixing is disabled to improve performance. May result invalid geometries.

Disponibilidade: Requer GDAL 1.7 ou superior.

Note
If there is a no data value set for a band, pixels with that value will not be returned except in the case of exclude_nodata_value=false.

Note
Se você se importa somente com pixeis contados com um dado valor em um raster, é mais rápido usar: ST_ValueCount.

Note
Isto é diferente da ST_PixelAsPolygons onde uma geometria retorna para cada pixel independente do valor do pixel.

Exemplos

```
-- this syntax requires PostgreSQL 9.3+
SELECT val, ST_AsText(geom) As geomwkt
FROM (
SELECT dp.*
FROM dummy_rast, LATERAL ST_DumpAsPolygons(rast) AS dp
WHERE rid = 2
) As foo
WHERE val BETWEEN 249 and 251
ORDER BY val;

 val |                                                  geomwkt
```

```
-----+-------------------------------------------------------------------------
 249 | POLYGON((3427927.95 5793243.95,3427927.95 5793243.85,3427928 5793243.85,
       3427928 5793243.95,3427927.95 5793243.95))
 250 | POLYGON((3427927.75 5793243.9,3427927.75 5793243.85,3427927.8 5793243.85,
       3427927.8 5793243.9,3427927.75 5793243.9))
 250 | POLYGON((3427927.8 5793243.8,3427927.8 5793243.75,3427927.85 5793243.75,
       3427927.85 5793243.8, 3427927.8 5793243.8))
 251 | POLYGON((3427927.75 5793243.85,3427927.75 5793243.8,3427927.8 5793243.8,
       3427927.8 5793243.85,3427927.75 5793243.85))
```

Veja também

geomval, ST_Value, ST_Polygon, ST_ValueCount

10.15.4 ST_Envelope

ST_Envelope — Retorna a representação de polígono da extensão do raster.

Synopsis

geometry **ST_Envelope**(raster rast);

Descrição

Retorna a representação de polígono da extensão do raster em unidades de coordenadas espaciais definidas pelo srid. É uma caixa delimitadora float8 mínima representada como um polígono.

O polígono é definido pelos pontos do canto da caixa delimitadora ((`MINX`, `MINY`), (`MINX`, `MAXY`), (`MAXX`, `MAXY`), (`MAXX`, `MINY`), (`MINX`, `MINY`))

Exemplos

```
SELECT rid, ST_AsText(ST_Envelope(rast)) As envgeomwkt
FROM dummy_rast;

 rid |                                        envgeomwkt
-----+-------------------------------------------------------------------------
   1 | POLYGON((0 0,20 0,20 60,0 60,0 0))
   2 | POLYGON((3427927 5793243,3427928 5793243,
       3427928 5793244,3427927 5793244, 3427927 5793243))
```

Veja também

ST_Envelope, ST_AsText, ST_SRID

10.15.5 ST_MinConvexHull

ST_MinConvexHull — Retorna a geometria de casco convexo do raster excluindo os pixeis SEM DADOS.

Synopsis

geometry **ST_MinConvexHull**(raster rast, integer nband=NULL);

Descrição

Retorna a geometria de casco convexo do raster excluindo os pixeis NODATA. Se `nband` for NULL, todas as bandas do raster serão consideradas.

Disponibilidade: 2.1.0

Exemplos

```
WITH foo AS (
    SELECT
        ST_SetValues(
            ST_SetValues(
                ST_AddBand(ST_AddBand(ST_MakeEmptyRaster(9, 9, 0, 0, 1, -1, 0, 0, 0), 1, '8 ↩
                    BUI', 0, 0), 2, '8BUI', 1, 0),
                1, 1, 1,
                ARRAY[
                    [0, 0, 0, 0, 0, 0, 0, 0, 0],
                    [0, 0, 0, 0, 0, 0, 0, 0, 0],
                    [0, 0, 0, 0, 0, 0, 0, 0, 0],
                    [0, 0, 0, 1, 0, 0, 0, 0, 1],
                    [0, 0, 0, 1, 1, 0, 0, 0, 0],
                    [0, 0, 0, 1, 0, 0, 0, 0, 0],
                    [0, 0, 0, 0, 0, 0, 0, 0, 0],
                    [0, 0, 0, 0, 0, 0, 0, 0, 0],
                    [0, 0, 0, 0, 0, 0, 0, 0, 0]
                ]::double precision[][]
            ),
            2, 1, 1,
            ARRAY[
                [0, 0, 0, 0, 0, 0, 0, 0, 0],
                [0, 0, 0, 0, 0, 0, 0, 0, 0],
                [0, 0, 0, 0, 0, 0, 0, 0, 0],
                [1, 0, 0, 0, 0, 1, 0, 0, 0],
                [0, 0, 0, 0, 1, 1, 0, 0, 0],
                [0, 0, 0, 0, 0, 1, 0, 0, 0],
                [0, 0, 0, 0, 0, 0, 0, 0, 0],
                [0, 0, 0, 0, 0, 0, 0, 0, 0],
                [0, 0, 1, 0, 0, 0, 0, 0, 0]
            ]::double precision[][]
        ) AS rast
)
SELECT
    ST_AsText(ST_ConvexHull(rast)) AS hull,
    ST_AsText(ST_MinConvexHull(rast)) AS mhull,
    ST_AsText(ST_MinConvexHull(rast, 1)) AS mhull_1,
    ST_AsText(ST_MinConvexHull(rast, 2)) AS mhull_2
FROM foo

                hull                |                 mhull                 |                ↩
                    mhull_1                |                mhull_2
-----------------------------------+---------------------------------------+---------------------------

 POLYGON((0 0,9 0,9 -9,0 -9,0 0)) | POLYGON((0 -3,9 -3,9 -9,0 -9,0 -3)) | POLYGON((3 -3,9  ↩
     -3,9 -6,3 -6,3 -3)) | POLYGON((0 -3,6 -3,6 -9,0 -9,0 -3))
```

Veja também

ST_Envelope, ST_ConvexHull, ST_ConvexHull, ST_AsText

10.15.6 ST_Polygon

ST_Polygon — Retorna um multipolígono formado pela união de pixeis que têm um valor que não é um valor sem dados. Se um número de banda for especificado, usa-se 1.

Synopsis

geometry **ST_Polygon**(raster rast, integer band_num=1);

Descrição

Changed 3.3.0, validation and fixing is disabled to improve performance. May result invalid geometries.

Disponibilidade: 0.1.6 Requer GDAL 1.7 ou superior.

Melhorias: 2.1.0 Velocidade aprimorada (fully C-Based) e o multipolígono que retorna é assegurado como válido.

Alterações: 2.1.0 Nas versões anteriores retornaria polígono, foi alterado para sempre voltar multipolígono.

Exemplos

```
-- by default no data band value is 0 or not set, so polygon will return a square polygon
SELECT ST_AsText(ST_Polygon(rast)) As geomwkt
FROM dummy_rast
WHERE rid = 2;

geomwkt
-----------------------------------------
MULTIPOLYGON(((3427927.75 5793244,3427928 5793244,3427928 5793243.75,3427927.75  ↩
    5793243.75,3427927.75 5793244)))


-- now we change the no data value of first band
UPDATE dummy_rast SET rast = ST_SetBandNoDataValue(rast,1,254)
WHERE rid = 2;
SELECt rid, ST_BandNoDataValue(rast)
from dummy_rast where rid = 2;

-- ST_Polygon excludes the pixel value 254 and returns a multipolygon
SELECT ST_AsText(ST_Polygon(rast)) As geomwkt
FROM dummy_rast
WHERE rid = 2;

geomwkt
-----------------------------------------------------
MULTIPOLYGON(((3427927.9 5793243.95,3427927.85 5793243.95,3427927.85 5793244,3427927.9  ↩
    5793244,3427927.9 5793243.95)),((3427928 5793243.85,3427928 5793243.8,3427927.95  ↩
    5793243.8,3427927.95 5793243.85,3427927.9 5793243.85,3427927.9 5793243.9,3427927.9  ↩
    5793243.95,3427927.95 5793243.95,3427928 5793243.95,3427928 5793243.85)),((3427927.8  ↩
    5793243.75,3427927.75 5793243.75,3427927.75 5793243.8,3427927.75 5793243.85,3427927.75  ↩
    5793243.9,3427927.75 5793244,3427927.8 5793244,3427927.8 5793243.9,3427927.8  ↩
    5793243.85,3427927.85 5793243.85,3427927.85 5793243.8,3427927.85 5793243.75,3427927.8  ↩
    5793243.75)))

-- Or if you want the no data value different for just one time

SELECT ST_AsText(
    ST_Polygon(
        ST_SetBandNoDataValue(rast,1,252)
        )
```

```
    ) As geomwkt
FROM dummy_rast
WHERE rid =2;

geomwkt
---------------------------------
MULTIPOLYGON(((3427928 5793243.85,3427928 5793243.8,3427928 5793243.75,3427927.85  ↩
    5793243.75,3427927.8 5793243.75,3427927.8 5793243.8,3427927.75 5793243.8,3427927.75  ↩
    5793243.85,3427927.75 5793243.9,3427927.75 5793244,3427927.8 5793244,3427927.85  ↩
    5793244,3427927.9 5793244,3427928 5793244,3427928 5793243.95,3427928 5793243.85) ↩
    ,(3427927.9 5793243.9,3427927.9 5793243.85,3427927.95 5793243.85,3427927.95  ↩
    5793243.9,3427927.9 5793243.9)))
```

Veja também

ST_Value, ST_DumpAsPolygons

10.16 Operadores Raster

10.16.1 &&

&& — Retorna VERDADE se a caixa limitadora de A intersecta a caixa limitadora de B.

Synopsis

boolean **&&**(raster A , raster B);
boolean **&&**(raster A , geometry B);
boolean **&&**(geometry B , raster A);

Descrição

O operador `&&` retorna `TRUE` se a caixa limitadora da geometria/raster A intersecta a caixa limitadora da geometria/raster B.

Note
Este operador fará uso de qualquer índice que pode estar disponível nos rasters.

Disponibilidade: 2.0.0

Exemplos

```
SELECT A.rid As a_rid, B.rid As b_rid, A.rast && B.rast As intersect
 FROM dummy_rast AS A CROSS JOIN dummy_rast AS B LIMIT 3;

 a_rid | b_rid | intersect
-------+-------+-----------
     2 |     2 | t
     2 |     3 | f
     2 |     1 | f
```

10.16.2 &<

&< — Retorna VERDADE se uma caixa limitadora de A está à esquerda da de B.

Synopsis

boolean **&<**(raster A , raster B);

Descrição

O operador &< retorna VERDADE se a caixa limitadora da geometria A sobrepõe ou está à esquerda da caixa da geometria B, ou mais precisamente, sobrepõe ou NÃO está à direita da caixa limitadora da geometria B.

Note
Este operador fará uso de qualquer índice que pode estar disponível nos rasters.

Exemplos

```
SELECT A.rid As a_rid, B.rid As b_rid, A.rast &< B.rast As overleft
 FROM dummy_rast AS A CROSS JOIN dummy_rast AS B;

a_rid | b_rid | overleft
------+-------+----------
    2 |     2 | t
    2 |     3 | f
    2 |     1 | f
    3 |     2 | t
    3 |     3 | t
    3 |     1 | f
    1 |     2 | t
    1 |     3 | t
    1 |     1 | t
```

10.16.3 &>

&> — Retorna VERDADE se uma caixa limitadora de A está à direita da de B.

Synopsis

boolean **&>**(raster A , raster B);

Descrição

O operador &> retorna TRUE se a caixa delimitadora do raster A sobrepuser ou estiver à direita da do raster B, ou mais precisamente, sobrepuser ou NÃO estiver à esquerda da do raster B.

Note
Este operador fará uso de qualquer um dos indexes que talvez estejam disponíveis nas geometrias.

Exemplos

```
SELECT A.rid As a_rid, B.rid As b_rid, A.rast &> B.rast As overright
 FROM dummy_rast AS A CROSS JOIN dummy_rast AS B;

 a_rid | b_rid | overright
-------+-------+-----------
     2 |     2 | t
     2 |     3 | t
     2 |     1 | t
     3 |     2 | f
     3 |     3 | t
     3 |     1 | f
     1 |     2 | f
     1 |     3 | t
     1 |     1 | t
```

10.16.4 =

= — Retorna VERDADEse a caixa limitadora de A for a mesma de B. Utiliza precisão dupla de caixa limitadora.

Synopsis

boolean =(raster A , raster B);

Descrição

O operador = retorna VERDADE se a caixa limitadora da geometria/geografia A é a mesma da de B. O PostgreSQL usa o operadores =, <, e > definidos para geometrias para representar ordens e comparações internas de geometrias (ex. em um GRUPO ou ORDEM por oração).

Caution
Este operador NÃO fará uso de nenhum índice que podem estar disponíveis nos rasters. Use ~=. Este operador existe em sua maioria para poder ser agrupado pela coluna raster.

Disponibilidade: 2.1.0

Veja também

~=

10.16.5 @

@ — Retorna VERDADEse a caixa limitadora de A estiver contida pela de B. Utiliza precisão dupla de caixa limitadora.

Synopsis

boolean @(raster A , raster B);
boolean @(geometry A , raster B);
boolean @(raster B , geometry A);

Descrição

O operador @ retorna TRUE se a caixa delimitadora do raster/geometria A estiver contida pela caixa delimitadora do raster/geometria B.

Note
Este operador usará índices espaciais nos rasters.

Disponibilidade: 2.0.0 raster @ raster, raster @ geometria introduzida

Disponibilidade: 2.0.5 geometria @ raster introduzida

Veja também

~

10.16.6 ~=

~= — Retorna VERDADE se a caixa limitadora de A é a mesma de B.

Synopsis

boolean **~=**(raster A , raster B);

Descrição

O operador ~= retorna TRUE se a caixa delimitadora do raster A for a mesma da do raster B.

Note
Este operador fará uso de qualquer índice que pode estar disponível nos rasters.

Disponibilidade: 2.0.0

Exemplos

Casos de uso muito úteis é pegar dois conjuntos de bandas raster únicas que são do mesmo pedaço, mas representam temas diferentes e criar uma multi banda raster

```
SELECT ST_AddBand(prec.rast, alt.rast) As new_rast
    FROM prec INNER JOIN alt ON (prec.rast ~= alt.rast);
```

Veja também

ST_AddBand, =

10.16.7 ~

~ — Retorna TRUE se a caixa delimitadora de A estiver contida na do B. Utiliza caixa delimitadora de precisão dupla.

Synopsis

boolean ~(raster A , raster B);
boolean ~(geometry A , raster B);
boolean ~(raster B , geometry A);

Descrição

O operador ~ retorna `TRUE` se a caixa delimitadora do raster/geometria A estiver contida na caixa delimitadora do raster/geometria B.

Note
Este operador usará índices espaciais nos rasters.

Disponibilidade: 2.0.0

Veja também

@

10.17 Relações raster e raster de banda espacial

10.17.1 ST_Contains

ST_Contains — Retorna verdade se nenhum ponto do raster rasteB estiver no exterior do raster rastA e pelo menos um ponto do interior do rastB estiver no interior do rastA.

Synopsis

boolean **ST_Contains**(raster rastA , integer nbandA , raster rastB , integer nbandB);
boolean **ST_Contains**(raster rastA , raster rastB);

Descrição

O raster rastA contém o rastB se e somente se nenhum ponto do rastB estiver no exterior do rastA. Se o número de banda não for fornecido (ou for NULL), apenas o casco convexo do raster será considerado no teste. Se o número de banda for fornecido, somente aqueles pixeis com valor (não NODATA) são considerados no teste.

Note
Esta função fará uso de qualquer índice que possa estar disponível nos rasters.

Note
Para testar a relação espacial de um raster e uma geometria, usa ST_Polygon no raster, ex.: ST_Contains(ST_Polygon(raster), geometria) ou ST_Contains(geometria, ST_Polygon(raster)).

Note
ST_Contains() é o inverso da ST_Within(). Logo, ST_Contains(rastA, rastB) implica ST_Within(rastB, rastA).

Disponibilidade: 2.1.0

Exemplos

```
-- specified band numbers
SELECT r1.rid, r2.rid, ST_Contains(r1.rast, 1, r2.rast, 1) FROM dummy_rast r1 CROSS JOIN  ↩
    dummy_rast r2 WHERE r1.rid = 1;

NOTICE:  The first raster provided has no bands
 rid | rid | st_contains
-----+-----+-------------
   1 |   1 |
   1 |   2 | f
```

```
-- no band numbers specified
SELECT r1.rid, r2.rid, ST_Contains(r1.rast, r2.rast) FROM dummy_rast r1 CROSS JOIN  ↩
    dummy_rast r2 WHERE r1.rid = 1;
 rid | rid | st_contains
-----+-----+-------------
   1 |   1 | t
   1 |   2 | f
```

Veja também

ST_Intersects, ST_Within

10.17.2 ST_ContainsProperly

ST_ContainsProperly — Retorna verdade se o rastB intersectar o interior do rastA, mas não o limite ou exterior do ratA.

Synopsis

boolean **ST_ContainsProperly**(raster rastA , integer nbandA , raster rastB , integer nbandB);
boolean **ST_ContainsProperly**(raster rastA , raster rastB);

Descrição

O raster rastA contém devidamente o rastB se ele intersectar o interior do rastA, mas não o limite ou exterior do rastA. Se o número de banda não for fornecido (ou for NULL), apenas o casco convexo do raster será considerado no teste. Se o número de banda for fornecido, somente aqueles pixeis com valor (não NODATA) serão considerados no teste.

O rastA não se contém devidamente, mas se contém.

Note
Esta função fará uso de qualquer índice que possa estar disponível nos rasters.

Note
Para testar a relação espacial de um raster e uma geometria, usa ST_Polygon no raster, ex.: ST_ContainsProperly(ST_Polygon(raster), geometria) ou ST_ContainsProperly(geometria, ST_Polygon(raster)).

Disponibilidade: 2.1.0

Exemplos

```
SELECT r1.rid, r2.rid, ST_ContainsProperly(r1.rast, 1, r2.rast, 1) FROM dummy_rast r1 CROSS ↩
     JOIN dummy_rast r2 WHERE r1.rid = 2;

 rid | rid | st_containsproperly
-----+-----+---------------------
   2 |   1 | f
   2 |   2 | f
```

Veja também

ST_Intersects, ST_Contains

10.17.3 ST_Covers

ST_Covers — Retorna verdade se nenhum ponto do rastB estiver de fora do rastA.

Synopsis

boolean **ST_Covers**(raster rastA , integer nbandA , raster rastB , integer nbandB);
boolean **ST_Covers**(raster rastA , raster rastB);

Descrição

O rastA cobre rastB se e somente se nenhum ponto do rastB estiver no exterior do rastA. Se o número de banda não for fornecido (ou for NULO), apenas o casco convexo do raster é considerado no teste. Se o número de banda for fornecido, apenas aqueles pixeis com valor (não NODATA) serão considerados no teste.

Note
Esta função fará uso de qualquer índice que possa estar disponível nos rasters.

Note
Para testar a relação espacial de um raster e uma geometria, usa ST_Polygon no raster, ex.: ST_Coveres(ST_Polygon(raster), geometria) ou ST_Coveres(geometria, ST_Polygon(raster)).

Disponibilidade: 2.1.0

Exemplos

```
SELECT r1.rid, r2.rid, ST_Covers(r1.rast, 1, r2.rast, 1) FROM dummy_rast r1 CROSS JOIN ↩
    dummy_rast r2 WHERE r1.rid = 2;

 rid | rid | st_covers
-----+-----+-----------
   2 |   1 | f
   2 |   2 | t
```

Veja também

ST_Intersects, ST_CoveredBy

10.17.4 ST_CoveredBy

ST_CoveredBy — Retorna verdade se nenhum ponto do rastA estiver de fora do rastB.

Synopsis

boolean **ST_CoveredBy**(raster rastA , integer nbandA , raster rastB , integer nbandB);
boolean **ST_CoveredBy**(raster rastA , raster rastB);

Descrição

O rastA está coberto pelo rastB se e somente se nenhum ponto do rastA estiver no exterior do rastB. Se o número de banda não for fornecido (ou for NULO), apenas o casco convexo do raster é considerado no teste. Se o número de banda for fornecido, apenas aqueles pixeis com valor (não NODATA) serão considerados no teste.

Note
Esta função fará uso de qualquer índice que possa estar disponível nos rasters.

Note
Para testar a relação espacial de um raster e uma geometria, usa ST_Polygon no raster, ex.: ST_CoveredBy(ST_Polygon(raster), geometria) ou ST_CoveredBy(geometria, ST_Polygon(raster)).

Disponibilidade: 2.1.0

Exemplos

```
SELECT r1.rid, r2.rid, ST_CoveredBy(r1.rast, 1, r2.rast, 1) FROM dummy_rast r1 CROSS JOIN ↩
    dummy_rast r2 WHERE r1.rid = 2;

 rid | rid | st_coveredby
-----+-----+--------------
   2 |   1 | f
   2 |   2 | t
```

Veja também

ST_Intersects, ST_Covers

10.17.5 ST_Disjoint

ST_Disjoint — Retorna verdade se raster rastA não intersectar espacialmente com o rastB.

Synopsis

boolean **ST_Disjoint**(raster rastA , integer nbandA , raster rastB , integer nbandB);
boolean **ST_Disjoint**(raster rastA , raster rastB);

Descrição

O rastA e rastB estarão disjuntos se eles não dividirem nenhum espaço. Se o número de banda não for fornecido (ou for NULL), apenas o casco convexo do raster é considerado no teste. Se o número de banda for fornecido, apenas aqueles pixeis com valor (não NODATA) serão considerados no teste.

Note
Esta função NÃO usa nenhum índice.

Note
Para testar a relação espacial de um raster e uma geometria, use ST_Polygon no raster, ex.: ST_Disjoint(ST_Polygon(raster), geometria).

Disponibilidade: 2.1.0

Exemplos

```
-- rid = 1 has no bands, hence the NOTICE and the NULL value for st_disjoint
SELECT r1.rid, r2.rid, ST_Disjoint(r1.rast, 1, r2.rast, 1) FROM dummy_rast r1 CROSS JOIN  ↩
    dummy_rast r2 WHERE r1.rid = 2;

NOTICE:  The second raster provided has no bands
 rid | rid | st_disjoint
-----+-----+-------------
   2 |   1 |
   2 |   2 | f
```

```
-- this time, without specifying band numbers
SELECT r1.rid, r2.rid, ST_Disjoint(r1.rast, r2.rast) FROM dummy_rast r1 CROSS JOIN  ↩
    dummy_rast r2 WHERE r1.rid = 2;

 rid | rid | st_disjoint
-----+-----+-------------
   2 |   1 | t
   2 |   2 | f
```

Veja também

ST_Intersects

10.17.6 ST_Intersects

ST_Intersects — Retorna verdade se o raster rastA intersectar espacialmente com o raster rastB.

Synopsis

boolean **ST_Intersects**(raster rastA , integer nbandA , raster rastB , integer nbandB);
boolean **ST_Intersects**(raster rastA , raster rastB);
boolean **ST_Intersects**(raster rast , integer nband , geometry geommin);
boolean **ST_Intersects**(raster rast , geometry geommin , integer nband=NULL);
boolean **ST_Intersects**(geometry geommin , raster rast , integer nband=NULL);

Descrição

Retorna verdade se o rastA se intersectar espacialmente com o rastB. Se o número de banda não for fornecido (ou for NULO), apenas o casco convexo do raster é considerado no teste. Se o número de banda for fornecido, apenas aqueles pixeis com valor (não NODATA) serão considerados no teste.

Note
Esta função fará uso de qualquer índice que possa estar disponível nos rasters.

Melhorias: 2.0.0 suporte para interseções raster/raster foi introduzido.

Warning
Alterações: 2.1.0 O comportamento das variantes ST_Intersects(raster, geometria) foi alterado para combinar com ST_Intersects(geometria, raster).

Exemplos

```
-- different bands of same raster
SELECT ST_Intersects(rast, 2, rast, 3) FROM dummy_rast WHERE rid = 2;

 st_intersects
---------------
 t
```

Veja também

ST_Intersection, ST_Disjoint

10.17.7 ST_Overlaps

ST_Overlaps — Retorna verdade se o raster rastA e rastB se intersectam, mas um deles não contém o outro completamente.

Synopsis

boolean **ST_Overlaps**(raster rastA , integer nbandA , raster rastB , integer nbandB);
boolean **ST_Overlaps**(raster rastA , raster rastB);

Descrição

Retorna verdade se o raster rastA tocar espacialmente o raster rastB. Isso significa que eles se intersectam, mas um não contém o outro completamente. Se o número banda não for fornecido (ou for NULO), apenas o casco convexo do raster é considerado no teste. Se o número de banda for fornecido, apenas os pixeis com valor (não NODATA) serão considerados no teste.

Note
Esta função fará uso de qualquer índice que possa estar disponível nos rasters.

Note
Para testar a relação espacial de um raster e uma geometria, use ST_Polygon no raster, ex.: ST_Overlaps(ST_Polygon(raster), geometria).

Disponibilidade: 2.1.0

Exemplos

```
-- comparing different bands of same raster
SELECT ST_Overlaps(rast, 1, rast, 2) FROM dummy_rast WHERE rid = 2;

 st_overlaps
-------------
 f
```

Veja também

ST_Intersects

10.17.8 ST_Touches

ST_Touches — Retorna verdade se o raster rastA e rastB têm pelo menos um ponto em comum, mas seus interiores não se intersectarem.

Synopsis

boolean **ST_Touches**(raster rastA , integer nbandA , raster rastB , integer nbandB);
boolean **ST_Touches**(raster rastA , raster rastB);

Descrição

Retorna verdade se o raster rastA tocar espacialmente o raster rastB. Isso significa que eles têm pelo menos um ponto em comum, mas seus interiores não se intersectam. Se o número banda não for fornecido (ou for NULO), apenas o casco convexo do raster é considerado no teste. Se o número de banda for fornecido, apenas os pixeis com valor (não NODATA) serão considerados no teste.

Note
Esta função fará uso de qualquer índice que possa estar disponível nos rasters.

Note
Para testar a relação espacial de um raster e uma geometria, use ST_Polygon no raster, ex.: ST_Touches(ST_Polygon(raster), geometria).

Disponibilidade: 2.1.0

Exemplos

```
SELECT r1.rid, r2.rid, ST_Touches(r1.rast, 1, r2.rast, 1) FROM dummy_rast r1 CROSS JOIN  ↩
    dummy_rast r2 WHERE r1.rid = 2;

 rid | rid | st_touches
-----+-----+------------
   2 |   1 | f
   2 |   2 | f
```

Veja também

ST_Intersects

10.17.9 ST_SameAlignment

ST_SameAlignment — Retorna verdade se os rasters têm a mesma inclinação, escala, referência espacial, e deslocamento (pixeis podem ser colocados na mesma grade sem cortar eles) e falso se eles não notificarem problemas detalhados.

Synopsis

boolean **ST_SameAlignment**(raster rastA , raster rastB);
boolean **ST_SameAlignment**(double precision ulx1 , double precision uly1 , double precision scalex1 , double precision scaley1 , double precision skewx1 , double precision skewy1 , double precision ulx2 , double precision uly2 , double precision scalex2 , double precision scaley2 , double precision skewx2 , double precision skewy2);
boolean **ST_SameAlignment**(raster set rastfield);

Descrição

Versão não agregada (variantes 1 e 2): Retorna verdade se dois rasters (fornecidos diretamente ou feitos usando os valores esquerdo superior, escala, inclinação ou srid) têm a mesma escala, inclinação, srid e pelo menos um de qualquer dos quatro cantos de pixel de um raster cair em algum canto da grade do outro raster. Retorna falso se eles não e um AVISO detalhando o problema de alinhamento.

Versão agregada (variante 3): De um conjunto de rasters, retorna verdade se todos os rasters no conjunto estiverem alinhados. A função ST_SameAlignment() é "agregada" na terminologia do PostgreSQL. Isso significa que ela opera nas linhas de dados, da mesma maneira que as funções SUM() e AVG() operam.

Disponibilidade: 2.0.0

Melhorias: 2.1.0 adição da variante agregada

Exemplos: Rasters

```
SELECT ST_SameAlignment(
    ST_MakeEmptyRaster(1, 1, 0, 0, 1, 1, 0, 0),
    ST_MakeEmptyRaster(1, 1, 0, 0, 1, 1, 0, 0)
) as sm;

sm
----
t
```

```
SELECT ST_SameAlignment(A.rast,b.rast)
 FROM dummy_rast AS A CROSS JOIN dummy_rast AS B;

 NOTICE:  The two rasters provided have different SRIDs
NOTICE:  The two rasters provided have different SRIDs
 st_samealignment
------------------
 t
 f
 f
 f
```

Veja também

Section 9.1, ST_NotSameAlignmentReason, ST_MakeEmptyRaster

10.17.10 ST_NotSameAlignmentReason

ST_NotSameAlignmentReason — Retorna a declaração de texto se os rasters estiverem alinhados e se não tiverem, uma razão do porquê.

Synopsis

text **ST_NotSameAlignmentReason**(raster rastA, raster rastB);

Descrição

Retorna a declaração de texto se os rasters estiverem alinhados e se não tiverem, uma razão do porquê.

Note
Se existem várias razões do porquê os rasters não estão alinhados, apenas uma razão (o primeiro teste a falhar) retornará.

Disponibilidade: 2.1.0

Exemplos

```
SELECT
    ST_SameAlignment(
        ST_MakeEmptyRaster(1, 1, 0, 0, 1, 1, 0, 0),
        ST_MakeEmptyRaster(1, 1, 0, 0, 1.1, 1.1, 0, 0)
    ),
    ST_NotSameAlignmentReason(
        ST_MakeEmptyRaster(1, 1, 0, 0, 1, 1, 0, 0),
        ST_MakeEmptyRaster(1, 1, 0, 0, 1.1, 1.1, 0, 0)
    )
;

 st_samealignment |            st_notsamealignmentreason
------------------+-------------------------------------------------
 f                | The rasters have different scales on the X axis
(1 row)
```

Veja também

Section 9.1, ST_SameAlignment

10.17.11 ST_Within

ST_Within — Retorna verdade se nenhum ponto do raster rastA estiver no exterior do raster rastB e pelo menos um ponto do interior do rastA estiver no interior do rastB.

Synopsis

boolean **ST_Within**(raster rastA , integer nbandA , raster rastB , integer nbandB);
boolean **ST_Within**(raster rastA , raster rastB);

Descrição

O raster rastA está dentro do rastB se e somente se nenhum ponto do rastA estiver no exterior do rastB e pelo menos um ponto do interior do rastA estiver no interior do rastB. Se o número de banda não for fornecido (ou for NULL), apenas o casco convexo do raster será considerado no teste. Se o número de banda for fornecido, somente aqueles pixeis com valor (não NODATA) são considerados no teste.

Note
Este operador fará uso de qualquer índice que pode estar disponível nos rasters.

Note
Para testar a relação espacial de um raster e uma geometria, usa ST_Polygon no raster, ex.: ST_Within(ST_Polygon(raster), geometria) ou ST_Within(geometria, ST_Polygon(raster)).

Note
ST_Within() é o inverso da ST_Contains(). Logo, ST_Within(rastA, rastB) implica ST_Contains(rastB, rastA).

Disponibilidade: 2.1.0

Exemplos

```
SELECT r1.rid, r2.rid, ST_Within(r1.rast, 1, r2.rast, 1) FROM dummy_rast r1 CROSS JOIN  ↩
    dummy_rast r2 WHERE r1.rid = 2;

 rid | rid | st_within
-----+-----+-----------
   2 |   1 | f
   2 |   2 | t
```

Veja também

ST_Intersects, ST_Contains, ST_DWithin, ST_DFullyWithin

10.17.12 ST_DWithin

ST_DWithin — Retorna verdade se os rasters rastA e rastB estiverem dentro da distância especificada de cada um.

Synopsis

boolean **ST_DWithin**(raster rastA , integer nbandA , raster rastB , integer nbandB , double precision distance_of_srid);
boolean **ST_DWithin**(raster rastA , raster rastB , double precision distance_of_srid);

Descrição

Retorna verdade se os rasters rastA e rastB estiverem dentro da distância especificada de cada um. Se o número de banda não for fornecido (ou for NULO), apenas o casco convexo do raster é considerado no teste. Se o número de banda for fornecido, apenas aqueles pixeis com valor (não NODATA) serão considerados no teste.

A distância é especificada em unidades definidas pelo sistema de referência espacial dos rasters. Para esta função fazer sentido, os rasters fonte devem ser ambos da mesma projeção de coordenada, tendo o mesmo SRID.

Note
Este operador fará uso de qualquer índice que pode estar disponível nos rasters.

Note
Para testar a relação espacial de um raster e uma geometria, use ST_Polygon no raster, ex.: ST_DWithin(ST_Polygon(raster), geometria).

Disponibilidade: 2.1.0

Exemplos

```
SELECT r1.rid, r2.rid, ST_DWithin(r1.rast, 1, r2.rast, 1, 3.14) FROM dummy_rast r1 CROSS ↩
    JOIN dummy_rast r2 WHERE r1.rid = 2;

 rid | rid | st_dwithin
-----+-----+------------
   2 |   1 | f
   2 |   2 | t
```

Veja também

ST_Within, ST_DFullyWithin

10.17.13 ST_DFullyWithin

ST_DFullyWithin — Retorna verdade se os rasters rastA e rastB estiverem completamente dentro da distância especificada de cada um.

Synopsis

boolean **ST_DFullyWithin**(raster rastA , integer nbandA , raster rastB , integer nbandB , double precision distance_of_srid);
boolean **ST_DFullyWithin**(raster rastA , raster rastB , double precision distance_of_srid);

Descrição

Retorna verdade se os rasters rastA e rastB estiverem completamente dentro da distância especificada de cada um. Se o número de banda não for fornecido (ou for NULO), apenas o casco convexo do raster é considerado no teste. Se o número de banda for fornecido, apenas aqueles pixeis com valor (não NODATA) serão considerados no teste.

A distância é especificada em unidades definidas pelo sistema de referência espacial dos rasters. Para esta função fazer sentido, os rasters fonte devem ser ambos da mesma projeção de coordenada, tendo o mesmo SRID.

Note
Este operador fará uso de qualquer índice que pode estar disponível nos rasters.

Note
Para testar a relação espacial de um raster e uma geometria, use ST_Polygon no raster, ex.: ST_DFullyWithin(ST_Polygon(raster), geometria).

Disponibilidade: 2.1.0

Exemplos

```
SELECT r1.rid, r2.rid, ST_DFullyWithin(r1.rast, 1, r2.rast, 1, 3.14) FROM dummy_rast r1  ↩
    CROSS JOIN dummy_rast r2 WHERE r1.rid = 2;

 rid | rid | st_dfullywithin
-----+-----+-----------------
   2 |   1 | f
   2 |   2 | t
```

Veja também

ST_Within, ST_DWithin

10.18 Raster Tips

10.18.1 Out-DB Rasters

10.18.1.1 Directory containing many files

When GDAL opens a file, GDAL eagerly scans the directory of that file to build a catalog of other files. If this directory contains many files (e.g. thousands, millions), opening that file becomes extremely slow (especially if that file happens to be on a network drive such as NFS).

To control this behavior, GDAL provides the following environment variable: GDAL_DISABLE_READDIR_ON_OPEN. Set `GDAL_DISABLE_READDIR_ON_OPEN` to `TRUE` to disable directory scanning.

In Ubuntu (and assuming you are using PostgreSQL's packages for Ubuntu), `GDAL_DISABLE_READDIR_ON_OPEN` can be set in */etc/postgresql/POSTGRESQL_VERSION/CLUSTER_NAME/environment* (where POSTGRESQL_VERSION is the version of PostgreSQL, e.g. 9.6 and CLUSTER_NAME is the name of the cluster, e.g. maindb). You can also set PostGIS environment variables here as well.

```
# environment variables for postmaster process
# This file has the same syntax as postgresql.conf:
#  VARIABLE = simple_value
#  VARIABLE2 = 'any value!'
# I. e. you need to enclose any value which does not only consist of letters,
# numbers, and '-', '_', '.' in single quotes. Shell commands are not
# evaluated.
POSTGIS_GDAL_ENABLED_DRIVERS = 'ENABLE_ALL'

POSTGIS_ENABLE_OUTDB_RASTERS = 1

GDAL_DISABLE_READDIR_ON_OPEN = 'TRUE'
```

10.18.1.2 Maximum Number of Open Files

The maximum number of open files permitted by Linux and PostgreSQL are typically conservative (typically 1024 open files per process) given the assumption that the system is consumed by human users. For Out-DB Rasters, a single valid query can easily exceed this limit (e.g. a dataset of 10 year's worth of rasters with one raster for each day containing minimum and maximum temperatures and we want to know the absolute min and max value for a pixel in that dataset).

The easiest change to make is the following PostgreSQL setting: max_files_per_process. The default is set to 1000, which is far too low for Out-DB Rasters. A safe starting value could be 65536 but this really depends on your datasets and the queries run against those datasets. This setting can only be made on server start and probably only in the PostgreSQL configuration file (e.g. */etc/postgresql/POSTGRESQL_VERSION/CLUSTER_NAME/postgresql.conf* in Ubuntu environments).

```
...
# - Kernel Resource Usage -

max_files_per_process = 65536           # min 25
                                        # (change requires restart)
...
```

The major change to make is the Linux kernel's open files limits. There are two parts to this:

- Maximum number of open files for the entire system
- Maximum number of open files per process

10.18.1.2.1 Maximum number of open files for the entire system

You can inspect the current maximum number of open files for the entire system with the following example:

```
$ sysctl -a | grep fs.file-max
fs.file-max = 131072
```

If the value returned is not large enough, add a file to */etc/sysctl.d/* as per the following example:

```
$ echo "fs.file-max = 6145324"
>
> /etc/sysctl.d/fs.conf

$ cat /etc/sysctl.d/fs.conf
fs.file-max = 6145324

$ sysctl -p --system
* Applying /etc/sysctl.d/fs.conf ...
fs.file-max = 2097152
* Applying /etc/sysctl.conf ...

$ sysctl -a | grep fs.file-max
fs.file-max = 6145324
```

10.18.1.2.2 Maximum number of open files per process

We need to increase the maximum number of open files per process for the PostgreSQL server processes.

To see what the current PostgreSQL service processes are using for maximum number of open files, do as per the following example (make sure to have PostgreSQL running):

```
$ ps aux | grep postgres
postgres 31713  0.0  0.4 179012 17564 pts/0    S    Dec26   0:03 /home/dustymugs/devel/ ↩
    postgresql/sandbox/10/usr/local/bin/postgres -D /home/dustymugs/devel/postgresql/sandbox ↩
    /10/pgdata
postgres 31716  0.0  0.8 179776 33632 ?        Ss   Dec26   0:01 postgres: checkpointer  ↩
    process
postgres 31717  0.0  0.2 179144  9416 ?        Ss   Dec26   0:05 postgres: writer process
postgres 31718  0.0  0.2 179012  8708 ?        Ss   Dec26   0:06 postgres: wal writer  ↩
    process
postgres 31719  0.0  0.1 179568  7252 ?        Ss   Dec26   0:03 postgres: autovacuum  ↩
    launcher process
postgres 31720  0.0  0.1  34228  4124 ?        Ss   Dec26   0:09 postgres: stats collector  ↩
    process
postgres 31721  0.0  0.1 179308  6052 ?        Ss   Dec26   0:00 postgres: bgworker:  ↩
    logical replication launcher
```

```
$ cat /proc/31718/limits
Limit                     Soft Limit           Hard Limit           Units
Max cpu time              unlimited            unlimited            seconds
Max file size             unlimited            unlimited            bytes
Max data size             unlimited            unlimited            bytes
Max stack size            8388608              unlimited            bytes
Max core file size        0                    unlimited            bytes
Max resident set          unlimited            unlimited            bytes
Max processes             15738                15738                processes
Max open files            1024                 4096                 files
Max locked memory         65536                65536                bytes
Max address space         unlimited            unlimited            bytes
Max file locks            unlimited            unlimited            locks
Max pending signals       15738                15738                signals
Max msgqueue size         819200               819200               bytes
Max nice priority         0                    0
Max realtime priority     0                    0
Max realtime timeout      unlimited            unlimited            us
```

In the example above, we inspected the open files limit for Process 31718. It doesn't matter which PostgreSQL process, any of them will do. The response we are interested in is *Max open files*.

We want to increase *Soft Limit* and *Hard Limit* of *Max open files* to be greater than the value we specified for the PostgreSQL setting `max_files_per_process`. In our example, we set `max_files_per_process` to 65536.

In Ubuntu (and assuming you are using PostgreSQL's packages for Ubuntu), the easiest way to change the *Soft Limit* and *Hard Limit* is to edit **/etc/init.d/postgresql** (SysV) or **/lib/systemd/system/postgresql*.service** (systemd).

Let's first address the SysV Ubuntu case where we add **ulimit -H -n 262144** and **ulimit -n 131072** to **/etc/init.d/postgresql**.

```
...
case "$1" in
    start|stop|restart|reload)
        if [ "$1" = "start" ]; then
            create_socket_directory
        fi
    if [ -z "`pg_lsclusters -h`" ]; then
        log_warning_msg 'No PostgreSQL clusters exist; see "man pg_createcluster"'
        exit 0
    fi

    ulimit -H -n 262144
    ulimit -n 131072

    for v in $versions; do
        $1 $v || EXIT=$?
    done
    exit ${EXIT:-0}
        ;;
    status)
...
```

Now to address the systemd Ubuntu case. We will add **LimitNOFILE=131072** to every **/lib/systemd/system/postgresql*.service** file in the **[Service]** section.

```
...
[Service]

LimitNOFILE=131072

...
```

```
[Install]
WantedBy=multi-user.target
...
```

After making the necessary systemd changes, make sure to reload the daemon

```
systemctl daemon-reload
```

Chapter 11

PostGIS Extras

This chapter documents features found in the extras folder of the PostGIS source tarballs and source repository. These are not always packaged with PostGIS binary releases, but are usually PL/pgSQL based or standard shell scripts that can be run as is.

11.1 Padronizador de endereço

Essa é uma forquilha do padronizador PAGC (código original para essa porção era Padronizador de endereço PAGC PostgreSQL).

O padronizador de endereços é uma única linha de análise sintática que pega um endereço de entrada e o normaliza baseado em um conjunto de regras armazenado em uma table e helper lex e gaz tables.

O código é construído em uma uńica biblioteca de extensão chamada `address_standardizer` a qual pode ser instalada com `CREATE EXTENSION address_standardizer;`. Juntamente com a extensão address_standardizer, uma extensão amostra de dados chamada `address_standardizer_data_us` é construída, a qual contém gaz, lex e regras tables para dados dos EUA. Essas extensões podem ser instaladas via: `CREATE EXTENSION address_standardizer_data_us;`

O código para esta extensão pode ser encontrado no PostGIS `extensions/address_standardizer` e está atualmente autocontido.

Para instruções de instalação consulte: Section 2.3.

11.1.1 Como o analisador sintático funciona

O analisador sintático funciona da direita para a esquerda observando primeiramente os macro elementos para CEP, estado/província, cidade e depois observando os micro elementos para determinar se estamos lidando com uma casa numerada em uma rua ou intersecção ou ponto de referência. Ele normalmente não procura pelo código ou nome do país, mas isso poderia ser introduzido no futuro.

Código do país Suposto de ser EUA ou CA com base em: CEP como EUA ou estado/província do Canadá como EUA ou Canadá outro EUA

Caixa postal/CEP Esses são reconhecidos utilizando expressões Perl compatíveis. Esses regexs estão atualmente no parseaddress-api.c e são relativamente fáceis de alterar, caso seja necessário.

Estado/província Esses são reconhecidos utilizando expressões Perl compatíveis. Esses regexs estão atualmente no parseaddress-api.c e são relativamente fáceis de alterar, caso seja necessário.

11.1.2 Tipos de padronizador de endereço

11.1.2.1 stdaddr

stdaddr — Um tipo composto que consiste nos elementos de um endereço. Este é o tipo de retorno para `standardize_address` função.

Descrição

Um tipo composto dos elementos de um endereço. Este é o tipo de retorno para standardize_address função. Algumas descrições para elementos são emprestadas de PAGC Postal Attributes.

Os números pegos denotam o número de referência da saída no mesa de regras.

 This method needs address_standardizer extension.

construindo é texto (token number `0`): Refere ao número da construção ou nome. Identificadores e tipos de construções unparsed. Normalmente em branco para a maioria dos endereços.

house_num é um texto (número token `1`): Este é o número da rua em uma rua. Exemplo *75* em `75 Rua State`.

predir é um texto (número token `2`): NOME DA RUA PRE-DIRECTIONAL como Norte, Sul, Leste, Oeste etc.

qual é um texto (número token `3`): NOME DA RUA PRE-MODIFIER Exemplo *VELHA* em `3715 ESTRADA VELHA 99`.

pré tipo é um texto (número token `4`): TIPO DE PREFIXO DA RUA

nome é um texto (número token `5`): NOME DA RUA

suftype é um texto (número token `6`): TIPO DE CORREIO DA RUA ex. R, Av, Cir. Um tipo de rua seguindo o nome raiz da rua. Exemplo *RUA* em `75 Rua State`.

sufdir é um texto (número token `7`): RUA POST-DIRECTIONAL Um modificador direcional que segue o nome da rua.. Exemplo *OESTE* em `3715 DÉDIMA AVENIDA OESTE`.

rota rural is text (token number `8`): RURAL ROUTE . Example *7* in `RR 7`.

extra é texto: informação extra como número de pisos.

cidade is text (token number `10`): Exemplo Boston.

estado is text (token number `11`): Exemplo `MASSACHUSETTS`

país is text (token number `12`): Exemplo `USA`

caixa postal é texto CÓDIGO POSTAL (CÓDIGO ZIP) (número token `13`): Exemplo `02109`

box is text POSTAL BOX NUMBER (token number `14 and 15`): Example `02109`

unidade é texto Número do apartamento ou Número da suíte (número token `17`): Exemplo *3B* em `APTO 3B`.

11.1.3 Mesas de padronizador de endereço

11.1.3.1 mesa de regras

mesa de regras — A mesa de regras contém um conjunto de regras que mapeia a sequência de tokens de entrada de endereço para a sequência de saída. Uma regra é definida como um conjunto de tokens de entrada seguido por -1 (terminator) seguido por conjunto de tokens de saída seguido por -1 seguido por um número que denota tipo de regra seguido por um ranking de regra.

Descrição

Uma tabel regras deve ter pelo menos as colunas a seguir, embora você tenha permissão para adicionar mais para seus usos pessoais.

id Chave primária da tabela

regra campo de texto indicando a regra. Detalhes em PAGC Registros da regra do padronizador de endereços.

Uma regra consiste em um conjunto de não negativos inteiros representando tokens de entrada, terminados por um -1, seguidos por um número igual de não negativos inteiros representando atributos postais, terminados por um -1, seguidos por um inteiro representando um tipo de regra, seguido por um inteiro representando o rank da regra. As regras são ranqueadas de 0 (menor) até 17 (maior).

Então por exemplo `2 0 2 22 3 -1 5 5 6 7 3 -1 2 6`mapeia para a sequência de tokens de saída *TYPE NUMBER TYPE DIRECT QUALIF* para a sequência de saída *STREET STREET SUFTYP SUFDIR QUALIF*. A regra é uma ARC_C regra de rank 6.

Números para tokens da saída correpondentes estão listados em stdaddr.

Tokens de entrada

Cada regra começa com um conjunto de tokens de entrada seguidos por um terminator`-1`. Tokens de entrada extraídos de PAGC Input Tokens estão como segue:

Tokens de entrada baseados na forma

AMPERS (13). O ampersand (&) é frequentemente utilizado para abreviar a palavra "e".

DASH (9). Um caractere de pontuação.

DOBRO (21). Uma sequência de duas letras. Normalmente utilizadas como identificadoras.

FRACT (25). Frações são usadas algumas vezes em números cívicos ou de unidade.

MISTURADO (23). Uma string alfanumérica que contém ambos: letras e dígitos. Usado por identificadores.

NÚMERO (0). Uma string de dígito.

ORD (15). Representações como Primeiro ou 1ro. Normalmente usada em nomes de ruas.

ORD (18). Uma única letra.

PALAVRA (1). Uma palavra é uma string de letras de tamanho aleatório. Uma única letra pode ser os dois uma ÚNICA e uma PALAVRA.

Tokens de entrada baseados na função

BOXH (14). Palavras usadas para indicar caixas do correio. Por exemplo *Caixa* ou *CO Caixa*.

BUILDH (19). Palavras usadas para indicar prédios ou condomínios, normalmente como um prefixo. Por exemplo: *Torre* em *Torre 7A*.

BUILDT (24). Palavras e abreviações usadas para indicar prédios ou complexos de prédios, normalmente como um sufixo. Por exemplo: *Shopping Center*.

DIRETO (22). Palavras usadas para indicar direções, por exemplo *Norte*.

MILHA (20). Palavras usadas para indicar endereços marco miliário.

RUA (6). Palavras e abreviações usadas para indicar estradas e ruas. Po exemplo: a *Interestadual* em *Interestadual 5*

RR (8). Palavras e abreviações usadas para indicar rotas rurais. *RR*.

TIPO (2). Palavras e abreviação usadas para indicar tipos de ruas. Por exemplo: *R* or *AV*.

UNITH (16). palavras e abreviação usada para indicar sub endereços.Poe exemplo, *APTO* ou *UNIDADE*.

Tokens de entrada de tipo postal

QUÍNTUPLO (28). Um número de 5 dígitos. Identifica um código Zip

QUÁDRUPLO (29). Um número de 4 dígitos. Identifica ZIP4.

PCH (27). Uma sequência de letra número letra de 3 caracteres. Identifica um FSA, os 3 primeiros caracteres de um código postal canadense.

PCT (26). Uma sequência de número letra número de 3 caracteres. Identifica um LDU, os 3 últimos caracteres de um código postal canadense.

Palavras vazias

PALAVRAS VAZIAS combinadas com PALAVRAS. Uma string de múltiplas PALAVRAs e PALAVRAs VAZIAS será representada por uma única PALAVRA token.

PALAVRA VAZIA (7). Uma palavra com pouca significância lexical que pode ser omitida na análise sintática. Por exemplo: *O*.

Tokens de saída

Depois do primeiro-1 (terminator), segue os tokens de saída e sua ordem, seguido por um terminator `-1`. Números para tokens de saída correspondentes estão listados em stdaddr. Que estão permitidos é dependente em um tipo de regra. Tokens de saída válidos para cada tipo de regra estão listados em the section called "Tipos de Regra e Classificação".

Tipos de Regra e Classificação

A parte final da regra é o tipo de regra que é denotado por um dos seguintes, seguido por uma regra rank. As regras são classificadas de 0 (menor) até 17 (maior).

MACRO_C

(token number = "**0**"). A classe de regras para as orações parsing MACRO como *PLACE STATE ZIP*

MACRO_C output tokens (excerpted from http://www.pagcgeo.org/docs/html/pagc-12.html#--r-typ--.

CIDADE (número token "10"). Exemplo "Albany"

ESTADO (número token "11"). Exemplo "NY"

NAÇÃO (número token "12"). Este atributo não é usado na maioria dos arquivos de referência. Exemplo "USA"

POSTAL (número token "13"). (SADS elements "ZIP CODE" , "PLUS 4"). Este atributo é usado para o US Zip e os códigos postais canadenses.

MICRO_C

(número token = "**1**"). A classe de regras para orações parsing full MICRO (such as House, street, sufdir, predir, pretyp, suftype, qualif) (ie ARC_C plus CIVIC_C). Essas regras não são usadas na construção da frase.

MICRO_C output tokens (excerpted from http://www.pagcgeo.org/docs/html/pagc-12.html#--r-typ--.

CASA é um texto (número token `1`): Este é o número da rua em uma rua. Exemplo *75* em `75 Rua State`.

predir é um texto (número token `2`): NOME DA RUA PRE-DIRECTIONAL como Norte, Sul, Leste, Oeste etc.

qual é um texto (número token `3`): NOME DA RUA PRE-MODIFIER Exemplo *VELHA* em `3715 ESTRADA VELHA 99`.

pré tipo é um texto (número token `4`): TIPO DE PREFIXO DA RUA

rua é um texto (número token `5`): NOME DA RUA

suftype é um texto (número token `6`): TIPO DE CORREIO DA RUA ex. R, Av, Cir. Um tipo de rua seguindo o nome raiz da rua. Exemplo *RUA* em `75 Rua State`.

sufdir é um texto (número token `7`): RUA POST-DIRECTIONAL Um modificador direcional que segue o nome da rua.. Exemplo *OESTE* em `3715 DÉDIMA AVENIDA OESTE`.

ARC_C

(número token = "**2**"). A calsse de regras para orações parsing MICRO, excluindo o atributo CASA. Como usa o mesmo conjunto de tokens de saída como MICRO_C menos o token CASA.

CIVIC_C

(número token = "**3**"). A classe de regras para parsing o atributo da CASA.

EXTRA_C

(número token = "**4**"). A classe de regras para atributos parsing EXTRA - atributos excluídos do geocoding. Essas regras não são usadas na fase de construção.

EXTRA_C output tokens (excerpted from http://www.pagcgeo.org/docs/html/pagc-12.html#--r-typ--.

BLDNG (token number `0`): Unparsed identificadores e tipos de construção.

BOXH (token number `14`): The **BOX** in `BOX 3B`

BOXT (token number `15`): The **3B** in `BOX 3B`

RR (token number `8`): The **RR** in `RR 7`

UNITH (token number `16`): The **APT** in `APT 3B`

UNITT (token number `17`): The **3B** in `APT 3B`

DESCONHECIDO (token number `9`): Uma saída senão não classificada.

11.1.3.2 lex table

lex table — Uma gaz table é usada para classificar entrada e associado alfanumérico que entram com (a) tokens de entrada (See the section called "Tokens de entrada") e (b) representações padronizadas.

Descrição

Uma lex (diminutivo para léxico) table é usada para classificar entrada alfanumérica e associar que entra com the section called "Tokens de entrada" e (b) representações padronizadas. Coisas que você encontrará nessas tables são `UM` mapeado para stdword: `1`.

Um lex tem pelo menos as colunas seguintes na table. Você talvez adicione

id Chave primária da tabela

seq inteiro: definição de número?

palavra texto: a palavra de entrada

stdword texto: a palavra substituta padronizada

token inteiro: o tipo de palavra ele é. Só se usado nesse contexto será substituído. Disponível em PAGC Tokens.

11.1.3.3 gaz table

gaz table — Uma gaz table é usada para padronizar nomes de lugares e associações que entram com (a) tokens de entrada (See the section called "Tokens de entrada") e (b) representações padronizadas.

Descrição

A gaz (short for gazeteer) table is used to standardize place names and associate that input with the section called "Tokens de entrada" and (b) standardized representations. For example if you are in US, you may load these with State Names and associated abbreviations.

Uma gaz table tem pelo menos as colunas a seguir na table. Você talvez adicione mais colunas para seus próprios propósitos.

id Chave primária da tabela

seq inteiro: definição do número? - identificador usado para aquela ocasião da palavra

palavra texto: a palavra de entrada

stdword texto: a palavra substituta padronizada

token inteiro: o tipo de palavra ele é. Só se usado nesse contexto será substituído. Disponível em PAGC Tokens.

11.1.4 Funções do padronizador de endereços

11.1.4.1 debug_standardize_address

debug_standardize_address — Returns a json formatted text listing the parse tokens and standardizations

Synopsis

text **debug_standardize_address**(text lextab, text gaztab, text rultab, text micro, text macro=NULL);

Descrição

This is a function for debugging address standardizer rules and lex/gaz mappings. It returns a json formatted text that includes the matching rules, mapping of tokens, and best standardized address stdaddr form of an input address utilizing lex table table name, gaz table, and mesa de regras table names and an address.

For single line addresses use just `micro`

For two line address A `micro` consisting of standard first line of postal address e.g. `house_num street`, and a macro consisting of standard postal second line of an address e.g `city, state postal_code country`.

Elements returned in the json document are

input_tokens For each word in the input address, returns the position of the word, token categorization of the word, and the standard word it is mapped to. Note that for some input words, you might get back multiple records because some inputs can be categorized as more than one thing.

rules The set of rules matching the input and the corresponding score for each. The first rule (highest scoring) is what is used for standardization

stdaddr The standardized address elements stdaddr that would be returned when running standardize_address

Availability: 3.4.0

 This method needs address_standardizer extension.

Exemplos

Using address_standardizer_data_us extension

```
CREATE EXTENSION address_standardizer_data_us; -- only needs to be done once
```

Variant 1: Single line address and returning the input tokens

```
SELECT it->
>'pos' AS position, it->
>'word' AS word, it->
>'stdword' AS standardized_word,
            it->
>'token' AS token, it->
>'token-code' AS token_code
    FROM jsonb(
            debug_standardize_address('us_lex',
                'us_gaz', 'us_rules', 'One Devonshire Place, PH 301, Boston, MA 02109')
                 ) AS s, jsonb_array_elements(s->'input_tokens') AS it;
```

```
position |    word    | standardized_word | token  | token_code
---------+------------+-------------------+--------+------------
 0       | ONE        | 1                 | NUMBER | 0
 0       | ONE        | 1                 | WORD   | 1
 1       | DEVONSHIRE | DEVONSHIRE        | WORD   | 1
 2       | PLACE      | PLACE             | TYPE   | 2
 3       | PH         | PATH              | TYPE   | 2
 3       | PH         | PENTHOUSE         | UNITT  | 17
 4       | 301        | 301               | NUMBER | 0
(7 rows)
```

Variant 2: Multi line address and returning first rule input mappings and score

```
SELECT (s->'rules'->0->
>'score')::numeric AS score, it->
>'pos' AS position,
        it->
>'input-word' AS word, it->
>'input-token' AS input_token, it->
>'mapped-word' AS standardized_word,
            it->
>'output-token' AS output_token
    FROM jsonb(
            debug_standardize_address('us_lex',
                'us_gaz', 'us_rules', 'One Devonshire Place, PH 301', 'Boston, MA 02109')
                 ) AS s, jsonb_array_elements(s->'rules'->0->'rule_tokens') AS it;
```

```
score    | position |    word    | input_token | standardized_word | output_token
---------+----------+------------+-------------+-------------------+--------------
 0.876250 | 0        | ONE        | NUMBER      | 1                 | HOUSE
 0.876250 | 1        | DEVONSHIRE | WORD        | DEVONSHIRE        | STREET
 0.876250 | 2        | PLACE      | TYPE        | PLACE             | SUFTYP
 0.876250 | 3        | PH         | UNITT       | PENTHOUSE         | UNITT
 0.876250 | 4        | 301        | NUMBER      | 301               | UNITT
(5 rows)
```

Veja também

stdaddr, mesa de regras, lex table, gaz table, Pagc_Normalize_Address

11.1.4.2 parse_address

parse_address — Pega um endereço linha 1 e quebra em partes

Synopsis

record **parse_address**(text address);

Descrição

Returns takes an address as input, and returns a record output consisting of fields *num*, *street*, *street2*, *address1*, *city*, *state*, *zip*, *zipplus*, *country*.

Disponibilidade: 2.2.0

 This method needs address_standardizer extension.

Exemplos

Endereços Únicos

```
SELECT num, street, city, zip, zipplus
        FROM parse_address('1 Devonshire Place, Boston, MA 02109-1234') AS a;
```

```
num |      street      |  city  |  zip  | zipplus
-----+------------------+--------+-------+---------
 1   | Devonshire Place | Boston | 02109 | 1234
```

Table de endereços

```
-- basic table
CREATE TABLE places(addid serial PRIMARY KEY, address text);

INSERT INTO places(address)
VALUES ('529 Main Street, Boston MA, 02129'),
 ('77 Massachusetts Avenue, Cambridge, MA 02139'),
 ('25 Wizard of Oz, Walaford, KS 99912323'),
 ('26 Capen Street, Medford, MA'),
 ('124 Mount Auburn St, Cambridge, Massachusetts 02138'),
 ('950 Main Street, Worcester, MA 01610');

 -- parse the addresses
 -- if you want all fields you can use (a).*
SELECT addid, (a).num, (a).street, (a).city, (a).state, (a).zip, (a).zipplus
FROM (SELECT addid, parse_address(address) As a
 FROM places) AS p;
```

```
addid | num |        street        |   city    | state |  zip  | zipplus
-------+-----+----------------------+-----------+-------+-------+---------
     1 | 529 | Main Street          | Boston    | MA    | 02129 |
     2 | 77  | Massachusetts Avenue | Cambridge | MA    | 02139 |
     3 | 25  | Wizard of Oz         | Walaford  | KS    | 99912 | 323
     4 | 26  | Capen Street         | Medford   | MA    |       |
     5 | 124 | Mount Auburn St      | Cambridge | MA    | 02138 |
     6 | 950 | Main Street          | Worcester | MA    | 01610 |
(6 rows)
```

Veja também

11.1.4.3 standardize_address

standardize_address — Retorna uma forma stdaddr de um endereço de entrada utilizando lex, gaz e rule tables.

Synopsis

stdaddr **standardize_address**(text lextab, text gaztab, text rultab, text address);
stdaddr **standardize_address**(text lextab, text gaztab, text rultab, text micro, text macro);

Descrição

Retorna a uma stdaddr forma de um endereço de entrada utilizando lex table table nome, gaz table, e mesa de regras table nomes e endereço.

Variante 1: Pega um endereço como uma única linha.

Variante 2: Pega o endereço em duas partes. Uma `micro` que consiste em padronizar a primeira linha do endereço postal ex. `house_num street`, e uma macro que consiste em adronizar a segunda linha de um endereço postal ex. `city, state postal_code country`.

Disponibilidade: 2.2.0

 This method needs address_standardizer extension.

Exemplos

Using address_standardizer_data_us extension

```
CREATE EXTENSION address_standardizer_data_us; -- only needs to be done once
```

Variante 1: Única linha de endereço. Isso não funcionou bem com os endereços não-EUA

```
SELECT house_num, name, suftype, city, country, state, unit  FROM standardize_address(' ↩
    us_lex',
                            'us_gaz', 'us_rules', 'One Devonshire Place, PH 301, Boston, MA  ↩
                               02109');
```

```
house_num |     name     | suftype |  city   | country |     state     |      unit
----------+--------------+---------+---------+---------+---------------+-----------------
1         | DEVONSHIRE | PLACE   | BOSTON | USA     | MASSACHUSETTS | # PENTHOUSE 301
```

Utilizando tables compactadas com o geocoder tiger. Este exemplo só funciona se você instalou `postgis_tiger_geocoder`.

```
SELECT *  FROM standardize_address('tiger.pagc_lex',
         'tiger.pagc_gaz', 'tiger.pagc_rules', 'One Devonshire Place, PH 301, Boston, MA  ↩
            02109-1234');
```

Para tornar a leitura mais fácil nós iremos abandonar a saída usando a extensão hstore CREATE EXTENSION hstore; você vai precisar instalar

```
SELECT (each(hstore(p))).*
 FROM standardize_address('tiger.pagc_lex', 'tiger.pagc_gaz',
   'tiger.pagc_rules', 'One Devonshire Place, PH 301, Boston, MA 02109') As p;
```

```
key     |      value
------------+-----------------
 box        |
 city       | BOSTON
 name       | DEVONSHIRE
 qual       |
 unit       | # PENTHOUSE 301
 extra      |
 state      | MA
 predir     |
 sufdir     |
 country    | USA
 pretype    |
 suftype    | PL
 building   |
 postcode   | 02109
 house_num  | 1
 ruralroute |
(16 rows)
```

variante 2: Como um endereço de duas partes

```
SELECT (each(hstore(p))).*
 FROM standardize_address('tiger.pagc_lex', 'tiger.pagc_gaz',
   'tiger.pagc_rules', 'One Devonshire Place, PH 301', 'Boston, MA 02109, US') As p;
```

```
key     |      value
------------+-----------------
 box        |
 city       | BOSTON
 name       | DEVONSHIRE
 qual       |
 unit       | # PENTHOUSE 301
 extra      |
 state      | MA
 predir     |
 sufdir     |
 country    | USA
 pretype    |
 suftype    | PL
 building   |
 postcode   | 02109
 house_num  | 1
 ruralroute |
(16 rows)
```

Veja também

stdaddr, mesa de regras, lex table, gaz table, Pagc_Normalize_Address

11.2 Tiger Geocoder

Existem outras fontes abertas geocoder para o PostGIS, que, ao contrário do tiger geocoder, têm a vantagem do suporte geocoding para muitos países

- Nominatim usa dados OpenStreetMap gazeteer formatados. Requer o osm2pgsql para carregar os dados, PostgreSQL 8.4+ e PostGIS 1.5+ para funcionar. É compactado como uma interface de serviço da web e parece ter sido criado para ser chamado

como webservice. Assim como o geocoder, ele tem dois componentes: o geocoder e o geocoder reverso. Na documentação não fica claro se ele tem uma interface SQL pura conforme o geocoder ou se tem um bom acordo da lógica implementado na interface da web.

- GIS Graphy também utiliza PostGIS e, como Nominatim, funciona com os dados OpenStreetMap (OSM). Ele possui um carregador para carregar dados OSM e, correspondente ao Nominatim, é capaz de geocoding não só nos EUA. Bem como Nominatim, ele executa como webservice e confia no Java 1.5, Servlet apps, Solr. O GisGraphy é uma multiplataforma e também possui um geocoder reverso juntamente com outros aspectos.

11.2.1 Drop_Indexes_Generate_Script

Drop_Indexes_Generate_Script — Gera uma script que derruba todas as chaves não primárias e indexes não únicos no esquema tiger e esquema especificado de usuário. Padroniza esquema para: `tiger_data` se nenhum esquema é especificado.

Synopsis

text **Drop_Indexes_Generate_Script**(text param_schema=tiger_data);

Descrição

Gera uma script que derruba todas as chaves não primárias e indexes não únicos no esquema tiger e esquema especificado de usuário. Padroniza esquema para: `tiger_data` se nenhum esquema é especificado.

Isso é útil para minimizar o excesso de indexes que pode confundir o organizador de pesquisas ou ocupar um espaço desnecessário. Use combinado com Install_Missing_Indexes, para adicionar os indexes usados pelo geocoder.

Disponibilidade: 2.0.0

Exemplos

```
SELECT drop_indexes_generate_script() As actionsql;
actionsql
-----------------------------------------------------
DROP INDEX tiger.idx_tiger_countysub_lookup_lower_name;
DROP INDEX tiger.idx_tiger_edges_countyfp;
DROP INDEX tiger.idx_tiger_faces_countyfp;
DROP INDEX tiger.tiger_place_the_geom_gist;
DROP INDEX tiger.tiger_edges_the_geom_gist;
DROP INDEX tiger.tiger_state_the_geom_gist;
DROP INDEX tiger.idx_tiger_addr_least_address;
DROP INDEX tiger.idx_tiger_addr_tlid;
DROP INDEX tiger.idx_tiger_addr_zip;
DROP INDEX tiger.idx_tiger_county_countyfp;
DROP INDEX tiger.idx_tiger_county_lookup_lower_name;
DROP INDEX tiger.idx_tiger_county_lookup_snd_name;
DROP INDEX tiger.idx_tiger_county_lower_name;
DROP INDEX tiger.idx_tiger_county_snd_name;
DROP INDEX tiger.idx_tiger_county_the_geom_gist;
DROP INDEX tiger.idx_tiger_countysub_lookup_snd_name;
DROP INDEX tiger.idx_tiger_cousub_countyfp;
DROP INDEX tiger.idx_tiger_cousub_cousubfp;
DROP INDEX tiger.idx_tiger_cousub_lower_name;
DROP INDEX tiger.idx_tiger_cousub_snd_name;
DROP INDEX tiger.idx_tiger_cousub_the_geom_gist;
DROP INDEX tiger_data.idx_tiger_data_ma_addr_least_address;
DROP INDEX tiger_data.idx_tiger_data_ma_addr_tlid;
DROP INDEX tiger_data.idx_tiger_data_ma_addr_zip;
DROP INDEX tiger_data.idx_tiger_data_ma_county_countyfp;
```

```
DROP INDEX tiger_data.idx_tiger_data_ma_county_lookup_lower_name;
DROP INDEX tiger_data.idx_tiger_data_ma_county_lookup_snd_name;
DROP INDEX tiger_data.idx_tiger_data_ma_county_lower_name;
DROP INDEX tiger_data.idx_tiger_data_ma_county_snd_name;
:
:
```

Veja Também

Install_Missing_Indexes, Missing_Indexes_Generate_Script

11.2.2 Drop_Nation_Tables_Generate_Script

Drop_Nation_Tables_Generate_Script — Gera uma script que derruba todas as tables no esquema específico que começa com `county_all`, `state_all` ou código de estado seguido por `condado` ou `estado`.

Synopsis

text **Drop_Nation_Tables_Generate_Script**(text param_schema=tiger_data);

Descrição

Gera uma script que derruba todas as tables no esquema específico que começa com `county_all`, `state_all` ou código de estado seguido por `condado` ou `estado`. Isso é necessário se você está atualizando os dados do `tiger_2010` para o `tiger_2011`.

Disponibilidade: 2.1.0

Exemplos

```
SELECT drop_nation_tables_generate_script();
DROP TABLE tiger_data.county_all;
DROP TABLE tiger_data.county_all_lookup;
DROP TABLE tiger_data.state_all;
DROP TABLE tiger_data.ma_county;
DROP TABLE tiger_data.ma_state;
```

Veja Também

Loader_Generate_Nation_Script

11.2.3 Drop_State_Tables_Generate_Script

Drop_State_Tables_Generate_Script — Gera uma script que derruba todas as tables no esquema específico que estão prefixados com abreviação do estado. Padroniza o esquema para `tiger_data` se nenhum esquema estiver especificado.

Synopsis

text **Drop_State_Tables_Generate_Script**(text param_state, text param_schema=tiger_data);

Descrição

Gera uma script que derruba todas as tables no esquema específico que estão prefixados com abreviação do estado. Padroniza o esquema para `tiger_data` se nenhum esquema estiver especificado. Essa função é útil para derrubar tables de um estado antes de recarregar um estado em caso de algo ter dado errado durante seu carregamento anterior.

Disponibilidade: 2.0.0

Exemplos

```
SELECT drop_state_tables_generate_script('PA');
DROP TABLE tiger_data.pa_addr;
DROP TABLE tiger_data.pa_county;
DROP TABLE tiger_data.pa_county_lookup;
DROP TABLE tiger_data.pa_cousub;
DROP TABLE tiger_data.pa_edges;
DROP TABLE tiger_data.pa_faces;
DROP TABLE tiger_data.pa_featnames;
DROP TABLE tiger_data.pa_place;
DROP TABLE tiger_data.pa_state;
DROP TABLE tiger_data.pa_zip_lookup_base;
DROP TABLE tiger_data.pa_zip_state;
DROP TABLE tiger_data.pa_zip_state_loc;
```

Veja Também

Loader_Generate_Script

11.2.4 Geocode

Geocode — Assimila um endereço como uma string (ou outro endereço normalizado) e gera um conjunto de localizações possíveis que inclui um ponto em NAD 83 long lat, um endereço normalizado para cada um e a avaliação. Quanto menor a avaliação, maior a chance de combinar. Os resultados são separados com menor avaliação em primeiro lugar. Pode passar no resultados máximos, até 10, e restrict_region (padrão NULO)

Synopsis

setof record **geocode**(varchar address, integer max_results=10, geometry restrict_region=NULL, norm_addy OUT addy, geometry OUT geomout, integer OUT rating);
setof record **geocode**(norm_addy in_addy, integer max_results=10, geometry restrict_region=NULL, norm_addy OUT addy, geometry OUT geomout, integer OUT rating);

Descrição

Assimila um endereço como uma string (ou endereço já normalizado) e gera uma série de possíveis localizações que inclui um ponto em NAD 83 long lat, um `normalized_address` (addy) para cada e a avaliação. Quanto menor a avaliação, maior a chance de combinar. Os resultados são separados com a menor avaliação em primeiro lugar. Usa os dados (limites, faces, addr) Tiger, uma string confusa PostgreSQL (soundex,levenshtein) linha de interpolação PostGIS para interpolar endereços ao longo dos limites do Tiger. Quanto maior a avaliação, menos o geocoder estará correto. O ponto geocodificado é padronizado para compensar 10 metros da lina central do lado (E/D) que o endereço da rua está localizado.

Melhorias: 2.0.0 para suportar o Tiger 2010, dados estruturados e lógica revisada para melhorar a velocidade, exatidão do geocoding e para compensar ponto da linha central para o lado do endereço que a rua está localizada. O novo parâmetro `max_results` é útil para especificar números dos melhores resultados ou apenas retornar o melhor resultado.

Exemplos: Básico

Os exemplos abaixo estão em um único processador 3.0 GHZ no Windows 7 com 2GB ram executando PostgreSQL 9.1rc1/PostGIS 2.0 carregados com todos os dados de estado Tiger MA,MN,CA, RI.

Combinações exatas são mais fáceis de computar (61ms)

```
SELECT g.rating, ST_X(g.geomout) As lon, ST_Y(g.geomout) As lat,
    (addy).address As stno, (addy).streetname As street,
    (addy).streettypeabbrev As styp, (addy).location As city, (addy).stateabbrev As st,( ↩
       addy).zip
    FROM geocode('75 State Street, Boston MA 02109', 1) As g;
 rating |        lon        |      lat       | stno | street | styp |  city  | st |  zip
--------+-------------------+----------------+------+--------+------+--------+----+-------
      0 | -71.0557505845646 | 42.35897920691 |   75 | State  | St   | Boston | MA | 02109
```

Mesmo se o zip não tiver passado no geocode pode estimar (demorou cerca de 122-150 ms)

```
SELECT g.rating, ST_AsText(ST_SnapToGrid(g.geomout,0.00001)) As wktlonlat,
    (addy).address As stno, (addy).streetname As street,
    (addy).streettypeabbrev As styp, (addy).location As city, (addy).stateabbrev As st,( ↩
       addy).zip
    FROM geocode('226 Hanover Street, Boston, MA',1) As g;
 rating |         wktlonlat         | stno | street  | styp |  city  | st |  zip
--------+---------------------------+------+---------+------+--------+----+-------
      1 | POINT(-71.05528 42.36316) |  226 | Hanover | St   | Boston | MA | 02113
```

Sabe lidar com erros de ortografia e fornece mair de uma possibilidade de solução com avaliações e tomadas maiores (500ms).

```
SELECT g.rating, ST_AsText(ST_SnapToGrid(g.geomout,0.00001)) As wktlonlat,
    (addy).address As stno, (addy).streetname As street,
    (addy).streettypeabbrev As styp, (addy).location As city, (addy).stateabbrev As st,( ↩
       addy).zip
    FROM geocode('31 - 37 Stewart Street, Boston, MA 02116',1) As g;
 rating |         wktlonlat         | stno | street | styp |  city  | st |  zip
--------+---------------------------+------+--------+------+--------+----+-------
     70 | POINT(-71.06466 42.35114) |   31 | Stuart | St   | Boston | MA | 02116
```

Utilizando para fazer um agrupamento geocode de endereços. Mais fácil para configurar `max_results=1`. Processa somente aqueles que ainda não foram geocodificados (não possuem avaliação).

```
CREATE TABLE addresses_to_geocode(addid serial PRIMARY KEY, address text,
        lon numeric, lat numeric, new_address text, rating integer);

INSERT INTO addresses_to_geocode(address)
VALUES ('529 Main Street, Boston MA, 02129'),
 ('77 Massachusetts Avenue, Cambridge, MA 02139'),
 ('25 Wizard of Oz, Walaford, KS 99912323'),
 ('26 Capen Street, Medford, MA'),
 ('124 Mount Auburn St, Cambridge, Massachusetts 02138'),
 ('950 Main Street, Worcester, MA 01610');

-- only update the first 3 addresses (323-704 ms -  there are caching and shared memory  ↩
    effects so first geocode you do is always slower) --
-- for large numbers of addresses you don't want to update all at once
-- since the whole geocode must commit at once
-- For this example we rejoin with LEFT JOIN
-- and set to rating to -1 rating if no match
-- to ensure we don't regeocode a bad address
UPDATE addresses_to_geocode
  SET  (rating, new_address, lon, lat)
    = ( COALESCE(g.rating,-1), pprint_addy(g.addy),
```

```
       ST_X(g.geomout)::numeric(8,5), ST_Y(g.geomout)::numeric(8,5) )
FROM (SELECT addid, address
    FROM addresses_to_geocode
    WHERE rating IS NULL ORDER BY addid LIMIT 3) As a
    LEFT JOIN LATERAL geocode(a.address,1) As g ON true
WHERE a.addid = addresses_to_geocode.addid;

result
-----
Query returned successfully: 3 rows affected, 480 ms execution time.

SELECT * FROM addresses_to_geocode WHERE rating is not null;

 addid |                   address                    |    lon    |   lat    |  ↩
                  new_address                | rating
-------+----------------------------------------------+-----------+----------+----------------------

     1 | 529 Main Street, Boston MA, 02129            | -71.07177 | 42.38357 | 529 Main St, ↩
         Boston, MA 02129             |      0
     2 | 77 Massachusetts Avenue, Cambridge, MA 02139 | -71.09396 | 42.35961 | 77  ↩
        Massachusetts Ave, Cambridge, MA 02139 |      0
     3 | 25 Wizard of Oz, Walaford, KS 99912323       | -97.92913 | 38.12717 | Willowbrook, ↩
         KS 67502                     |    108
(3 rows)
```

Exemplos: Usando Filtros Geométricos

```
SELECT g.rating, ST_AsText(ST_SnapToGrid(g.geomout,0.00001)) As wktlonlat,
    (addy).address As stno, (addy).streetname As street,
    (addy).streettypeabbrev As styp,
    (addy).location As city, (addy).stateabbrev As st,(addy).zip
  FROM geocode('100 Federal Street, MA',
        3,
        (SELECT ST_Union(the_geom)
            FROM place WHERE statefp = '25' AND name = 'Lynn')::geometry
        ) As g;

 rating |         wktlonlat         | stno | street  | styp | city | st |  zip
--------+---------------------------+------+---------+------+------+----+-------
      7 | POINT(-70.96796 42.4659)  |  100 | Federal | St   | Lynn | MA | 01905
     16 | POINT(-70.96786 42.46853) | NULL | Federal | St   | Lynn | MA | 01905
(2 rows)

Time: 622.939 ms
```

Veja Também

Normalize_Address, Pprint_Addy, ST_AsText, ST_SnapToGrid, ST_X, ST_Y

11.2.5 Geocode_Intersection

Geocode_Intersection — Assimila 2 ruas que se intersectam e um estado, cidade, zip, e gera um conjunto de possíveis localizações no primeiro cruzamento que está na intersecção, também inclui um geomout como o ponto de localização em NAD 83 long lat, um `normalized_address` (addy) para cada localização, e a avaliação. Quanto menor a avaliação, maior a chance de combinar. Os resultados são separados com menor avaliação em primeiro lugar. Pode passar nos resultados máximos, até 10. Usa dados Tiger (limites, faces, addr), string confusa do PostgreSQL (soundex, evenshtein).

Synopsis

setof record **geocode_intersection**(text roadway1, text roadway2, text in_state, text in_city, text in_zip, integer max_results=10, norm_addy OUT addy, geometry OUT geomout, integer OUT rating);

Descrição

Assimila 2 ruas que se intersectam e um estado, cidade, zip, e gera um conjunto de possíveis localizações no primeiro cruzamento que está na intersecção, também inclui um geomout como o ponto de localização em NAD 83 long lat para cada localização, e a avaliação. Quanto menor a avaliação, maior a chance de combinar. Os resultados são separados com menor avaliação em primeiro lugar. Pode passar nos resultados máximos, até 10. Retorna `normalized_address` (addy) para cada, geomout como o ponto da localização em nad 83 long lat, and the rating. Quanto menor a avaliação, maior a chance de combinar. Os resultados são separados com menor avaliação em primeiro lugar. Usa dados Tiger (limites, faces, addr), string confusa do PostgreSQL (soundex, evenshtein).

Disponibilidade: 2.0.0

Exemplos: Básico

Os exemplos abaixo estão em um único processador 3.0 GHZ no Windows 7 com 2GB ram executando PostgreSQL 9.0/PostGIS 1.5 carregados com todos os dados de estado MA Tiger carregados. Atualmente um pouco devagar (3000 ms)

Testando no Windows 2003 64-bit 8GB on PostGIS 2.0 PostgreSQL 64-bit Tiger 2011 dados carregados -- (41ms)

```
SELECT pprint_addy(addy), st_astext(geomout),rating
            FROM geocode_intersection( 'Haverford St','Germania St', 'MA', 'Boston',  ↩
                '02130',1);
           pprint_addy            |         st_astext          | rating
----------------------------------+----------------------------+--------
98 Haverford St, Boston, MA 02130 | POINT(-71.101375 42.31376) |      0
```

Mesmo se o zip não passar no geocoder pode estimar (demorou cerca de 3500 ms na caixa do windows 7), no o windows 2003 64-bit 741 ms

```
SELECT pprint_addy(addy), st_astext(geomout),rating
                FROM geocode_intersection('Weld', 'School', 'MA', 'Boston');
          pprint_addy          |        st_astext         | rating
-------------------------------+--------------------------+--------
 98 Weld Ave, Boston, MA 02119 | POINT(-71.099 42.314234) |      3
 99 Weld Ave, Boston, MA 02119 | POINT(-71.099 42.314234) |      3
```

Veja Também

Geocode, Pprint_Addy, ST_AsText

11.2.6 Get_Geocode_Setting

Get_Geocode_Setting — Retorna a configuração de valor específico armazenada na table tiger.geocode_settings.

Synopsis

text **Get_Geocode_Setting**(text setting_name);

Descrição

Retorna valor da configuração específica armazenada na table tiger.geocode_settings. As configurações te permitem comutar depuração de funções. Planos futuros serão para controlar a avaliação com as configurações. A seguir, a lista atual de configurações:

```
name              | setting |  unit   | category  |  ↩
                                                        short_desc
-------------------------------+---------+---------+-----------+-----------------------------------

 debug_geocode_address          | false   | boolean | debug     | outputs debug information ↩
     in notice log such as queries when geocode_address is called if true
 debug_geocode_intersection     | false   | boolean | debug     | outputs debug information ↩
     in notice log such as queries when geocode_intersection is called if true
 debug_normalize_address        | false   | boolean | debug     | outputs debug information ↩
     in notice log such as queries and intermediate expressions when normalize_address is  ↩
    called if true
 debug_reverse_geocode          | false   | boolean | debug     | if true, outputs debug  ↩
    information in notice log such as queries and intermediate expressions when  ↩
    reverse_geocode
 reverse_geocode_numbered_roads | 0       | integer | rating    | For state and county  ↩
    highways, 0 - no preference in name,
                                                                   1 - prefer the numbered  ↩
                                                                       highway name, 2 -  ↩
                                                                       prefer local state/ ↩
                                                                       county name
 use_pagc_address_parser        | false   | boolean | normalize | If set to true, will try  ↩
    to use the address_standardizer extension (via pagc_normalize_address)
                                                                  instead of tiger  ↩
                                                                      normalize_address built ↩
                                                                       one
```

Alterações: 2.2.0 : configurações padrão são guardadas em uma table chamada geocode_settings_default. As configurações personalizadas estão em geocode_settings e só contém aquelas que foram configuradas pelo usuário.

Disponibilidade: 2.1.0

Exemplo da configuração de retornar depuração

```
SELECT get_geocode_setting('debug_geocode_address) As result;
result
---------
false
```

Veja Também

Set_Geocode_Setting

11.2.7 Get_Tract

Get_Tract — Retorna o trecho ou campo de uma tract table onde a geometria está localizada. Padrão para retornar um nome curto para o trecho.

Synopsis

text **get_tract**(geometry loc_geom, text output_field=name);

Descrição

Uma dada geometria irá retornar o trecho da localização do censo daquela geometria. NAD 83 long lat é assumida se nenhum spatial ref sys estiver especificado.

Note

This function uses the census `tract` which is not loaded by default. If you have already loaded your state table, you can load tract as well as bg, and tabblock using the Loader_Generate_Census_Script script.
If you have not loaded your state data yet and want these additional tables loaded, do the following

```
UPDATE tiger.loader_lookuptables SET load = true WHERE load = false AND lookup_name
    IN('tract', 'bg', 'tabblock');
```

then they will be included by the Loader_Generate_Script.

Disponibilidade: 2.0.0

Exemplos: Básico

```
SELECT get_tract(ST_Point(-71.101375, 42.31376) ) As tract_name;
tract_name
---------
1203.01
```

```
--this one returns the tiger geoid
SELECT get_tract(ST_Point(-71.101375, 42.31376), 'tract_id' ) As tract_id;
tract_id
---------
25025120301
```

Veja Também

Geocode >

11.2.8 Install_Missing_Indexes

Install_Missing_Indexes — Encontra todas as tables com colunas chave usadas no ingresso geocoder e condições de filtros que estão perdendo os indexes usados nessas colunas e irão adicionar elas.

Synopsis

boolean **Install_Missing_Indexes**();

Descrição

Encontra todas as tables nos esquemas `tiger` e `tiger_data` com as colunas chave usadas no ingresso e filtros do geocoder que estão perdendo indexes nessas colunas e irão gerar o SQL DDL para definir o index para aquelas tables e, então, executar a script gerada. Essa é uma função ajudante, que adiciona novos indexes necessários para pesquisas mais rápidas que podem ter sido perdidas durante o carregamento. Essa função é uma acompanhante para Missing_Indexes_Generate_Script, que somada à script que cria index, também a executa. Ela é uma parte da script de atualização `update_geocode.sql`.

Disponibilidade: 2.0.0

Exemplos

```
SELECT install_missing_indexes();
         install_missing_indexes
-------------------------
 t
```

Veja Também

Loader_Generate_Script, Missing_Indexes_Generate_Script

11.2.9 Loader_Generate_Census_Script

Loader_Generate_Census_Script — Gera uma shell script para a plataforma específica para os estados que irão baixar o trecho do censo de estado Tiger, bg e dados de tables tabblocks, arranjar e carregar dentro do esquema `tiger_data`. Cada state script retornou como um relato separado.

Synopsis

setof text **loader_generate_census_script**(text[] param_states, text os);

Descrição

Gera uma shell script para a plataforma específica para os estados que irão baixar `tract` do censo de estado Tiger, block groups `bg` e dados de tables `tabblocks`, arranja e carrega dentro do esquema `tiger_data`. Cada state script retornou como um relato separado.

Utiliza unzip no Linux (7-zip no Windows por padrão) e wget para fazer o download. Usa Section 4.7.2 para carregar nos dados. Note que a menor unidade que ele faz é um estado inteiro. Ele só irá processar os arquivos nas pastas representativas e temporárias.

Isso usa as seguintes tables de controle para controlar o processo e diferentes variações de sintaxe OS shell.

1. `loader_variables` armazena pistas de várias variáveis como o site do censo, ano, dados e esquemas representativos.
2. `loader_platform` perfis de numerosas plataformas e onde as várias executáveis estão localizadas. Está com o windows e linux. Mais pode ser adicionado.
3. `loader_lookuptables` cada relato define um tipo de table (estado, condado), quer para processar relatos nelas ou para carregar eles. Define os passos para importar dados, dados de representação, adicionar, remove colunas, indexes e restrições para cada um. Cada table é prefixada com o estado de uma table em um esquema tiger. ex: cria `tiger_data.ma_faces`, os quais herda das `tiger.faces`

Disponibilidade: 2.0.0

Note
Loader_Generate_Script inclui essa lógica, mas se você instalou o geocoder tiger antes para o PostGIS 2.0.0 alpha5, você vai precisar executar esse nos estados que já fez para pegar essas tables adicionais.

Exemplos

Gerar script para carregar dados para selecionar estados no formato script shell do Windows.

```
SELECT loader_generate_census_script(ARRAY['MA'], 'windows');
-- result --
set STATEDIR="\gisdata\www2.census.gov\geo\pvs\tiger2010st\25_Massachusetts"
set TMPDIR=\gisdata\temp\
set UNZIPTOOL="C:\Program Files\7-Zip\7z.exe"
set WGETTOOL="C:\wget\wget.exe"
set PGBIN=C:\projects\pg\pg91win\bin\
set PGPORT=5432
set PGHOST=localhost
set PGUSER=postgres
set PGPASSWORD=yourpasswordhere
set PGDATABASE=tiger_postgis20
set PSQL="%PGBIN%psql"
set SHP2PGSQL="%PGBIN%shp2pgsql"
cd \gisdata

%WGETTOOL% http://www2.census.gov/geo/pvs/tiger2010st/25_Massachusetts/25/ --no-parent -- ↩
    relative --accept=*bg10.zip,*tract10.zip,*tabblock10.zip --mirror --reject=html
del %TMPDIR%\*.* /Q
%PSQL% -c "DROP SCHEMA tiger_staging CASCADE;"
%PSQL% -c "CREATE SCHEMA tiger_staging;"
cd %STATEDIR%
for /r %%z in (*.zip) do %UNZIPTOOL% e %%z  -o%TMPDIR%
cd %TMPDIR%
%PSQL% -c "CREATE TABLE tiger_data.MA_tract(CONSTRAINT pk_MA_tract PRIMARY KEY (tract_id) ) ↩
     INHERITS(tiger.tract); "
%SHP2PGSQL% -c -s 4269 -g the_geom   -W "latin1" tl_2010_25_tract10.dbf tiger_staging. ↩
    ma_tract10 | %PSQL%
%PSQL% -c "ALTER TABLE tiger_staging.MA_tract10 RENAME geoid10 TO tract_id;  SELECT  ↩
    loader_load_staged_data(lower('MA_tract10'), lower('MA_tract')); "
%PSQL% -c "CREATE INDEX tiger_data_MA_tract_the_geom_gist ON tiger_data.MA_tract USING gist ↩
    (the_geom);"
%PSQL% -c "VACUUM ANALYZE tiger_data.MA_tract;"
%PSQL% -c "ALTER TABLE tiger_data.MA_tract ADD CONSTRAINT chk_statefp CHECK (statefp =  ↩
    '25');"
:
```

Gerar script sh

```
STATEDIR="/gisdata/www2.census.gov/geo/pvs/tiger2010st/25_Massachusetts"
TMPDIR="/gisdata/temp/"
UNZIPTOOL=unzip
WGETTOOL="/usr/bin/wget"
export PGBIN=/usr/pgsql-9.0/bin
export PGPORT=5432
export PGHOST=localhost
export PGUSER=postgres
export PGPASSWORD=yourpasswordhere
export PGDATABASE=geocoder
PSQL=${PGBIN}/psql
SHP2PGSQL=${PGBIN}/shp2pgsql
cd /gisdata

wget http://www2.census.gov/geo/pvs/tiger2010st/25_Massachusetts/25/ --no-parent --relative ↩
     --accept=*bg10.zip,*tract10.zip,*tabblock10.zip --mirror --reject=html
rm -f ${TMPDIR}/*.*
${PSQL} -c "DROP SCHEMA tiger_staging CASCADE;"
${PSQL} -c "CREATE SCHEMA tiger_staging;"
```

```
cd $STATEDIR
for z in *.zip; do $UNZIPTOOL -o -d $TMPDIR $z; done
:
:
```

Veja Também

Loader_Generate_Script

11.2.10 Loader_Generate_Script

Loader_Generate_Script — Gera uma shell script para a plataforma específica para os estados que irão baixar dados Tiger, arranjar e carregar dentro do esquema `tiger_data`. Cada state script retorna como um registro separado. A versão mais nova suporta mudanças estruturais do Tiger 2010 e também carrega trecho do censo, block groups, e block tables.

Synopsis

setof text **loader_generate_script**(text[] param_states, text os);

Descrição

Gera uma shell script para a plataforma específica para os estados que irão baixar dados do Tiger, arranjar e carregar dentro do esquema `tiger_data`. Cada state script retorna como um registro separado.

Utiliza unzip no Linux (7-zip no Windows por padrão) e wget para fazer o download. Usa Section 4.7.2 para carregar nos dados. Note que a menor unidade que ele faz é um estado inteiro, mas você pode sobrescrever baixando os arquivos por conta própria. Ele só irá processar os arquivos nas pastas representativas e temporárias.

Isso usa as seguintes tables de controle para controlar o processo e diferentes variações de sintaxe OS shell.

1. `loader_variables` armazena pistas de várias variáveis como o site do censo, ano, dados e esquemas representativos.
2. `loader_platform` perfis de numerosas plataformas e onde as várias executáveis estão localizadas. Está com o windows e linux. Mais pode ser adicionado.
3. `loader_lookuptables` cada relato define um tipo de table (estado, condado), quer para processar relatos nelas ou para carregar eles. Define os passos para importar dados, dados de representação, adicionar, remove colunas, indexes e restrições para cada um. Cada table é prefixada com o estado de uma table em um esquema tiger. ex: cria `tiger_data.ma_faces`, os quais herda das `tiger.faces`

Disponibilidade: 2.0.0 para suportar tiger 2010 dados estruturados e carrega trecho (trecho) do censo , block groups (bg), e block (tabblocks) tables.

Note
If you are using pgAdmin 3, be warned that by default pgAdmin 3 truncates long text. To fix, change *File -> Options -> Query Tool -> Query Editor - > Max. characters per column* to larger than 50000 characters.

Exemplos

Using psql where gistest is your database and `/gisdata/data_load.sh` is the file to create with the shell commands to run.

```
psql -U postgres -h localhost -d gistest -A -t \
 -c "SELECT Loader_Generate_Script(ARRAY['MA'], 'gistest')"
> /gisdata/data_load.sh;
```

Gerar script para carregar dados para 2 estados na script de formato shell do Windows.

```
SELECT loader_generate_script(ARRAY['MA','RI'], 'windows') AS result;
-- result --
set TMPDIR=\gisdata\temp\
set UNZIPTOOL="C:\Program Files\7-Zip\7z.exe"
set WGETTOOL="C:\wget\wget.exe"
set PGBIN=C:\Program Files\PostgreSQL\9.4\bin\
set PGPORT=5432
set PGHOST=localhost
set PGUSER=postgres
set PGPASSWORD=yourpasswordhere
set PGDATABASE=geocoder
set PSQL="%PGBIN%psql"
set SHP2PGSQL="%PGBIN%shp2pgsql"
cd \gisdata

cd \gisdata
%WGETTOOL% ftp://ftp2.census.gov/geo/tiger/TIGER2015/PLACE/tl_*_25_* --no-parent --relative ↩
     --recursive --level=2 --accept=zip --mirror --reject=html
cd \gisdata/ftp2.census.gov/geo/tiger/TIGER2015/PLACE
:
:
```

Gerar script sh

```
SELECT loader_generate_script(ARRAY['MA','RI'], 'sh') AS result;
-- result --
TMPDIR="/gisdata/temp/"
UNZIPTOOL=unzip
WGETTOOL="/usr/bin/wget"
export PGBIN=/usr/lib/postgresql/9.4/bin
-- variables used by psql: https://www.postgresql.org/docs/current/static/libpq-envars.html
export PGPORT=5432
export PGHOST=localhost
export PGUSER=postgres
export PGPASSWORD=yourpasswordhere
export PGDATABASE=geocoder
PSQL=${PGBIN}/psql
SHP2PGSQL=${PGBIN}/shp2pgsql
cd /gisdata

cd /gisdata
wget ftp://ftp2.census.gov/geo/tiger/TIGER2015/PLACE/tl_*_25_* --no-parent --relative -- ↩
    recursive --level=2 --accept=zip --mirror --reject=html
cd /gisdata/ftp2.census.gov/geo/tiger/TIGER2015/PLACE
rm -f ${TMPDIR}/*.*
:
:
```

Veja Também

Section 2.4.1, Pprint_Addy, ST_AsText

11.2.11 Loader_Generate_Nation_Script

Loader_Generate_Nation_Script — Gerar uma script shell para a plataforma especificada que carrega as lookup tables de condado e estado.

Synopsis

text **loader_generate_nation_script**(text os);

Descrição

Gera uma script shell para a plataforma especificada que carrega as tables `county_all`, `county_all_lookup`, `state_all` dento do esquema `tiger_data`. Elas herdam respectivamente das tables `county`, `county_lookup`, `state` no esquema `tiger`.

Utiliza unzip no Linux (7-zip no Windows por padrão) e wget para fazer o download. Usa Section 4.7.2 para carregar nos dados.

Utiliza as seguintes tables de controle: `tiger.loader_platform`, `tiger.loader_variables`, e `tiger.loader_lookup` para controlar o processo e diferentes variações de sintaxe OS shell.

1. `loader_variables` armazena pistas de várias variáveis como o site do censo, ano, dados e esquemas representativos.
2. `loader_platform` perfis de numerosas plataformas e onde as várias executáveis estão localizadas. Está com o windows e linux/unix. Mais pode ser adicionado.
3. `loader_lookuptables` cada relato define um tipo de table (estado, condado), quer para processar relatos nelas ou para carregar eles. Define os passos para importar dados, dados de representação, adicionar, remove colunas, indexes e restrições para cada um. Cada table é prefixada com o estado de uma table em um esquema tiger. ex: cria `tiger_data.ma_faces`, os quais herda das `tiger.faces`

Enhanced: 2.4.1 zip code 5 tabulation area (zcta5) load step was fixed and when enabled, zcta5 data is loaded as a single table called zcta5_all as part of the nation script load.

Disponibilidade: 2.1.0

Note
If you want zip code 5 tabulation area (zcta5) to be included in your nation script load, do the following:

```
UPDATE tiger.loader_lookuptables SET load = true WHERE table_name = 'zcta510';
```

Note
Se você estiver executando a versão `tiger_2010` e quer recarregar como estado com `tiger_2011`, você vai precisar, para o primeiro carregamento, gerar e executar drop statements Drop_Nation_Tables_Generate_Script antes de executar essa script.

Exemplos

Gerar script para carregar dados de uma nação no Windows.

```
SELECT loader_generate_nation_script('windows');
```

Gerar script para carregar dados para os sistemas Linux/Unix.

```
SELECT loader_generate_nation_script('sh');
```

Veja Também

Loader_Generate_Script, Missing_Indexes_Generate_Script

11.2.12 Missing_Indexes_Generate_Script

Missing_Indexes_Generate_Script — Encontra todas as tables com colunas chave usadas no ingresso geocoder que estão perdendo indexes nessas colunas e irão gerar o SQL DDL para definir o index para essas tables.

Synopsis

text **Missing_Indexes_Generate_Script**();

Descrição

Encontra todas as tables nos esquemas `tiger` e `tiger_data` com as colunas chave usadas no ingresso geocoder que está perdendo indexes nessas colunas e irão gerar SQL DDL para definir o index para essas tables. Essa é uma função ajudante que adiciona novos indexes necessários para pesquisas mais rápidas que podem ter sido perdidas no carregamento. Assim como o geocoder é melhorado, essa função será atualizada para acomodar novos indexes que estão sendo usados. Se essa função não gera nada, significa que suas tables possuem o que achamos ser os indexes chave no lugar certo.

Disponibilidade: 2.0.0

Exemplos

```
SELECT missing_indexes_generate_script();
-- output: This was run on a database that was created before many corrections were made to ↩
     the loading script ---
CREATE INDEX idx_tiger_county_countyfp ON tiger.county USING btree(countyfp);
CREATE INDEX idx_tiger_cousub_countyfp ON tiger.cousub USING btree(countyfp);
CREATE INDEX idx_tiger_edges_tfidr ON tiger.edges USING btree(tfidr);
CREATE INDEX idx_tiger_edges_tfidl ON tiger.edges USING btree(tfidl);
CREATE INDEX idx_tiger_zip_lookup_all_zip ON tiger.zip_lookup_all USING btree(zip);
CREATE INDEX idx_tiger_data_ma_county_countyfp ON tiger_data.ma_county USING btree(countyfp ↩
    );
CREATE INDEX idx_tiger_data_ma_cousub_countyfp ON tiger_data.ma_cousub USING btree(countyfp ↩
    );
CREATE INDEX idx_tiger_data_ma_edges_countyfp ON tiger_data.ma_edges USING btree(countyfp);
CREATE INDEX idx_tiger_data_ma_faces_countyfp ON tiger_data.ma_faces USING btree(countyfp);
```

Veja Também

Loader_Generate_Script, Install_Missing_Indexes

11.2.13 Normalize_Address

Normalize_Address — Dado um endereço em texto de uma rua, retorna um tipo composto `norm_addy` que não tem um sufixo, prefixo e tipo padronizado, rua, nome de rua etc. quebrado e, campos separados. Essa função irá funcionar com os dados lookup compactados com o tiger_geocoder (dados do censo tiger não são necessários).

Synopsis

norm_addy **normalize_address**(varchar in_address);

Descrição

Dado um endereço em texto de uma rua, retorna um tipo composto `norm_addy` que não tem um sufixo, prefixo e tipo padronizado, rua, nome de rua etc. quebrado e, campos separados. Esse é o primeiro passo no processo de geocodificação para tornar todos os endereços normalizados no formato postal. Nenhum outro dado é requerido à parte do que está compactado com o geocoder.

Essa função utiliza as várias lookup tables direção/estado/sufixo pré carregadas com o tiger_geocoder e localizadas no esquema `tiger`, então, você não precisa baixar os dados do censo tiger ou qualquer outro tipo de dados para utilizá-la. Talvez você ache necessário adicionar mais abreviações ou nomes alternativos para as lookup tables no esquema `tiger`.

Utiliza várias tables lookup de controle localizadas no esquema `tiger` para normalizar o endereço de entrada.

Campos no tipo de objeto `norm_addy` retornou pela função nessa ordem, onde () indica um campo requerido pelo geocoder, [] indica um campo opcional:

(address) [predirAbbrev] (streetName) [streetTypeAbbrev] [postdirAbbrev] [internal] [location] [stateAbbrev] [zip] [parsed] [zip4] [address_alphanumeric]

Enhanced: 2.4.0 norm_addy object includes additional fields zip4 and address_alphanumeric.

1. `address` é um inteiro: O número da rua
2. `predirAbbrev` is varchar: Prefixo direcional para rua como N, S, L, O etc. Esse são controlados usando a table `direction_lookup`.
3. `streetName` varchar
4. A versão varchar `streetTypeAbbrev` abreviada dos tipos de rua: ex: St., Av., Cir. Elas são controladas usando a tables `street_type_lookup`.
5. As direções varchar `postdirAbbrev` abreviadas N, S, L, O etc. Elas são controladas utilizando a table `direction_lookup`.
6. Varchar `interno` endereço interno como um apartamento ou número de suíte.
7. Varchar `localização` normalmente uma cidade ou província governante.
8. Os estados varchar `stateAbbrev` dos EUA de dois caracteres. ex: MA, NY, MI. Estes são controlados pela table `state_lookup`.
9. `zip` varchar 5-digit zipcode. e.g. 02109.
10. `parsed` booleana - indica se um endereço foi formado pelo processo normalizador. A função normalize_address coloca isso como verdade antes de retornar o endereço.
11. `zip4` last 4 digits of a 9 digit zip code. Availability: PostGIS 2.4.0.
12. `address_alphanumeric` Full street number even if it has alpha characters like 17R. Parsing of this is better using Pagc_Normalize_Address function. Availability: PostGIS 2.4.0.

Exemplos

Gerar campos selecionados. Use Pprint_Addy se você quer uma saída textual.

```
SELECT address As orig, (g.na).streetname, (g.na).streettypeabbrev
 FROM (SELECT address, normalize_address(address) As na
        FROM addresses_to_geocode) As g;

                        orig                          |  streetname   | streettypeabbrev
------------------------------------------------------+---------------+------------------
 28 Capen Street, Medford, MA                         | Capen         | St
 124 Mount Auburn St, Cambridge, Massachusetts 02138  | Mount Auburn  | St
 950 Main Street, Worcester, MA 01610                 | Main          | St
 529 Main Street, Boston MA, 02129                    | Main          | St
 77 Massachusetts Avenue, Cambridge, MA 02139         | Massachusetts | Ave
 25 Wizard of Oz, Walaford, KS 99912323               | Wizard of Oz  |
```

Veja Também

Geocode, Pprint_Addy

11.2.14 Pagc_Normalize_Address

Pagc_Normalize_Address — Dado um endereço em texto de uma rua, retorna um tipo composto `norm_addy` que não tem um sufixo, prefixo e tipo padronizado, rua, nome de rua etc. quebrado e, campos separados. Essa função irá funcionar com os dados lookup compactados com o tiger_geocoder (dados do censo tiger não são necessários). Requer a extensão address_standardizer.

Synopsis

norm_addy **pagc_normalize_address**(varchar in_address);

Descrição

Dado um endereço em texto de uma rua, retorna um tipo composto `norm_addy` que não tem um sufixo, prefixo e tipo padronizado, rua, nome de rua etc. quebrado e, campos separados. Esse é o primeiro passo no processo de geocodificação para tornar todos os endereços normalizados no formato postal. Nenhum outro dado é requerido à parte do que está compactado com o geocoder.

Essa função utiliza as várias lookup tables pagc_* pré carregadas com o tiger_geocoder e localizadas no esquema `tiger`, então, você não precisa baixar os dados do censo tiger ou qualquer outro tipo de dados para utilizá-la. Talvez você ache necessário adicionar mais abreviações ou nomes alternativos para as lookup tables no esquema `tiger`.

Utiliza várias tables lookup de controle localizadas no esquema `tiger` para normalizar o endereço de entrada.

Campos no tipo de objeto `norm_addy` retornou pela função nessa ordem, onde () indica um campo requerido pelo geocoder, [] indica um campo opcional:

Existem pequenas variações no revestimento e formatação do Normalize_Address.

Disponibilidade: 2.1.0

 This method needs address_standardizer extension.

(address) [predirAbbrev] (streetName) [streetTypeAbbrev] [postdirAbbrev] [internal] [location] [stateAbbrev] [zip]

O standardaddr natural da extensão address_standardizer é um pouco mais rico que norm_addy, já que foi desenvolvido para suportar endereços internacionais (incluindo países). Os campos equivalentes do standardaddr são:

house_num,predir, name, suftype, sufdir, unit, city, state, postcode

Enhanced: 2.4.0 norm_addy object includes additional fields zip4 and address_alphanumeric.

1. `address` é um inteiro: O número da rua
2. `predirAbbrev` is varchar: Prefixo direcional para rua como N, S, L, O etc. Esse são controlados usando a table `direction_lookup`.
3. `streetName` varchar
4. A versão varchar `streetTypeAbbrev` abreviada dos tipos de rua: ex: St., Av., Cir. Elas são controladas usando a tables `street_type_lookup`.
5. As direções varchar `postdirAbbrev` abreviadas N, S, L, O etc. Elas são controladas utilizando a table `direction_lookup`.
6. Varchar `interno` endereço interno como um apartamento ou número de suíte.
7. Varchar `localização` normalmente uma cidade ou província governante.

8. Os estados varchar `stateAbbrev` dos EUA de dois caracteres. ex: MA, NY, MI. Estes são controlados pela table `state_lookup`.
9. `zip` varchar 5-digit zipcode. e.g. 02109.
10. `parsed` booleana - indica se um endereço foi formado pelo processo normalizador. A função normalize_address coloca isso como verdade antes de retornar o endereço.
11. `zip4` last 4 digits of a 9 digit zip code. Availability: PostGIS 2.4.0.
12. `address_alphanumeric` Full street number even if it has alpha characters like 17R. Parsing of this is better using Pagc_Normalize_Address function. Availability: PostGIS 2.4.0.

Exemplos

Exemplo de chamada única

```
SELECT addy.*
FROM pagc_normalize_address('9000 E ROO ST STE 999, Springfield, CO') AS addy;


 address | predirabbrev | streetname | streettypeabbrev | postdirabbrev | internal  |   ↩
    location    | stateabbrev | zip | parsed
---------+--------------+------------+------------------+---------------+-----------+-------------+-

    9000 | E            | ROO        | ST               |               | SUITE 999 |  ↩
       SPRINGFIELD | CO          |     | t
```

Batch call. Existem issues de velocidade com a forma que o postgis_tiger_geocoder empacota o address_standardizer. Elas serão solucionadas em edições posteriores. Para funcionar em volta delas, se você precisa de velocidade para geocodificação agrupada para gerar um normaddy em modo agrupado, você é instigado a usar a função address_standardizer standardize_address diretamente, como é mostrado abaixo, que é similar ao exercício que fizemos em Normalize_Address que usa os dados criados em Geocode.

```
WITH g AS (SELECT address, ROW((sa).house_num, (sa).predir, (sa).name
  , (sa).suftype, (sa).sufdir, (sa).unit , (sa).city, (sa).state, (sa).postcode, true):: ↩
     norm_addy As na
 FROM (SELECT address, standardize_address('tiger.pagc_lex'
       , 'tiger.pagc_gaz'
       , 'tiger.pagc_rules', address) As sa
        FROM addresses_to_geocode) As g)
SELECT address As orig, (g.na).streetname, (g.na).streettypeabbrev
 FROM  g;

 orig                                                 |  streetname   | streettypeabbrev
------------------------------------------------------+---------------+------------------
 529 Main Street, Boston MA, 02129                    | MAIN          | ST
 77 Massachusetts Avenue, Cambridge, MA 02139         | MASSACHUSETTS | AVE
 25 Wizard of Oz, Walaford, KS 99912323               | WIZARD OF     |
 26 Capen Street, Medford, MA                         | CAPEN         | ST
 124 Mount Auburn St, Cambridge, Massachusetts 02138  | MOUNT AUBURN  | ST
 950 Main Street, Worcester, MA 01610                 | MAIN          | ST
```

Veja Também

Normalize_Address, Geocode

11.2.15 Pprint_Addy

Pprint_Addy — Dado um objeto de tipo composto `norm_addy`, retorna uma representação impressa dele. Normalmente, usado em conjunto com o normalize_address.

Synopsis

varchar **pprint_addy**(norm_addy in_addy);

Descrição

Dado um objeto de tipo composto `norm_addy`, retorna uma representação impressa dele. Não é necessário nenhum outro tipo de dados além do que estão compactados com o geocoder.

Usado, normalmente, em conjunto com Normalize_Address.

Exemplos

Pretty print a single address

```
SELECT pprint_addy(normalize_address('202 East Fremont Street, Las Vegas, Nevada 89101'))  ↩
    As pretty_address;
            pretty_address
---------------------------------------
 202 E Fremont St, Las Vegas, NV 89101
```

Pretty print address a table of addresses

```
SELECT address As orig, pprint_addy(normalize_address(address)) As pretty_address
        FROM addresses_to_geocode;

                        orig                          |              pretty_address
------------------------------------------------------+------------------------------------------- ↩

 529 Main Street, Boston MA, 02129                    | 529 Main St, Boston MA, 02129
 77 Massachusetts Avenue, Cambridge, MA 02139         | 77 Massachusetts Ave, Cambridge, MA  ↩
    02139
 28 Capen Street, Medford, MA                         | 28 Capen St, Medford, MA
 124 Mount Auburn St, Cambridge, Massachusetts 02138  | 124 Mount Auburn St, Cambridge, MA  ↩
    02138
 950 Main Street, Worcester, MA 01610                 | 950 Main St, Worcester, MA 01610
```

Veja Também

Normalize_Address

11.2.16 Reverse_Geocode

Reverse_Geocode — Pega um ponto em um sistema de referência espacial conehcido e retorna um relato que contém um banco de dados de, teoricamente, possíveis endereços e um banco de dados de ruas cruzadas. Se include_strnum_range = verdade, inclui o alcance da rua nas ruas cruzadas.

Synopsis

record **Reverse_Geocode**(geometry pt, boolean include_strnum_range=false, geometry[] OUT intpt, norm_addy[] OUT addy, varchar[] OUT street);

Descrição

Pega um ponto em um sistema de referência espacial conehcido e retorna um relato que contém um banco de dados de, teoricamente, possíveis endereços e um banco de dados de ruas cruzadas. Se include_strnum_range = verdade, inclui o alcance da rua nas ruas cruzadas. include_strnum_range se torna falso se não passar. Os endereços são separados de acordo com qual rua um ponto é mais próximo, então, o primeiro endereço é o mais certo.

Porque dizemos endereços hipotéticos em vez de reais. Os dados Tiger não possuem endereços reais, somente variedades de ruas. O suposto endereço é interpolado baseado na variedade de ruas, por exemplo. Bem como interpolar um dos meus endereços de retorno em 26 Court St. and 26 Court Sq., sendo que não existem tais endereços. Isso se dá porque um ponto pode estar na esquina de 2 ruas e assim a lógica interpola as duas ruas. A lógica também presume que os endereços são espaçados igualmente ao longo de uma rua, o que está errado já que você pode ter um edifício municipal ocupando boa parte de uma rua, enquanto o resto das construções estão todas agrupadas no fim dela.

Nota: Esta função confia nos dados Tiger. Se você não carregou dados cobrindo a região deste ponto, então, você terá que ter um relato cheio de NULOS.

Elementos que retornaram do relato são como segue:

1. `intpt` é um arranjo de pontos: Estes são os pontos da linha central na rua mais próxima ao ponto de entrada. Existem vários pontos, assim com existem muitos endereços.
2. `addy` é um arranjo de norm_addy (endereços normalizados): Estes são arranjos de possíveis endereços que se encaixam no ponto de entrada. O primeiro no arranjo é o que mais se encaixa. Geralmente, deveria ter apenas um, exceto no caso de um ponto que esteja na esquina de 2 ou 3 ruas, ou do ponto que esteja em algum lugar na estrada e não ao lado da rua.
3. `street` um arranjo de varchar: Estes são ruas cruzadas (ou a rua) (ruas que interseccionam ou são as ruas que o ponto está projetado).

Enhanced: 2.4.1 if optional zcta5 dataset is loaded, the reverse_geocode function can resolve to state and zip even if the specific state data is not loaded. Refer to Loader_Generate_Nation_Script for details on loading zcta5 data.

Disponibilidade: 2.0.0

Exemplos

Exemplo de um ponto na esquina de duas ruas, mas mais perto de uma delas. É uma localização de MIT: 77 Massachusetts Ave, Cambridge, MA 02139. Note que mesmo não tendo nenhuma das 3 ruas, o PostgreSQL só retornará nulo para entradas acima do nosso limite mais alto, então, é seguro usar. Inclui variedades de ruas

```
SELECT pprint_addy(r.addy[1]) As st1, pprint_addy(r.addy[2]) As st2, pprint_addy(r.addy[3]) ↩
    As st3,
           array_to_string(r.street, ',') As cross_streets
       FROM reverse_geocode(ST_GeomFromText('POINT(-71.093902 42.359446)',4269),true) As r ↩
           ;

 result
 ------
     st1                                    | st2 | st3 |               cross_streets
--------------------------------------------+-----+-----+-------------------------------------------

 67 Massachusetts Ave, Cambridge, MA 02139 |     |     | 67 - 127 Massachusetts Ave,32 - 88 ↩
     Vassar St
```

Aqui, nós escolhemos não incluir as variedades de endereços para as ruas cruzadas e escolhemos uma localização realmente próxima de uma esquina de 2 ruas, assim, pode ser conhecido por dois endereços diferentes.

```
SELECT pprint_addy(r.addy[1]) As st1, pprint_addy(r.addy[2]) As st2,
pprint_addy(r.addy[3]) As st3, array_to_string(r.street, ',') As cross_str
FROM reverse_geocode(ST_GeomFromText('POINT(-71.06941 42.34225)',4269)) As r;
```

```
result
--------
               st1               |               st2                | st3 | cross_str
---------------------------------+----------------------------------+-----+------------------------ ←

 5 Bradford St, Boston, MA 02118 | 49 Waltham St, Boston, MA 02118 |     | Waltham St
```

Para este, nó reutilizamos nosso exemplo geocodificado de Geocode e só queremos o endereço primário e no máximo 2 ruas cruzadas.

```
SELECT actual_addr, lon, lat, pprint_addy((rg).addy[1]) As int_addr1,
    (rg).street[1] As cross1, (rg).street[2] As cross2
FROM (SELECT address As actual_addr, lon, lat,
    reverse_geocode( ST_SetSRID(ST_Point(lon,lat),4326) ) As rg
    FROM addresses_to_geocode WHERE rating
> -1) As foo;

                     actual_addr                      |    lon    |   lat    | ↩
                                int_addr1                 |     cross1      |    ↩
                         cross2
----------------------------------------------------------+-----------+----------+-----------------------

 529 Main Street, Boston MA, 02129                        | -71.07181 | 42.38359 | 527 Main St,  ↩
     Boston, MA 02129             | Medford St      |
 77 Massachusetts Avenue, Cambridge, MA 02139             | -71.09428 | 42.35988 | 77  ↩
     Massachusetts Ave, Cambridge, MA 02139 | Vassar St       |
 26 Capen Street, Medford, MA                             | -71.12377 | 42.41101 | 9 Edison Ave, ↩
      Medford, MA 02155           | Capen St        | Tesla Ave
 124 Mount Auburn St, Cambridge, Massachusetts 02138 | -71.12304 | 42.37328 | 3 University  ↩
     Rd, Cambridge, MA 02138      | Mount Auburn St |
 950 Main Street, Worcester, MA 01610                     | -71.82368 | 42.24956 | 3 Maywood St, ↩
      Worcester, MA 01603         | Main St         | Maywood Pl
```

Veja Também

Pprint_Addy, Pprint_Addy, ST_AsText

11.2.17 Topology_Load_Tiger

Topology_Load_Tiger — Carrega uma região definida de dados tiger em uma Topologia PostGIS e transforma os dados tiger para referência espacial da topologia e rompe para a tolerância precisa da topologia.

Synopsis

text **Topology_Load_Tiger**(varchar topo_name, varchar region_type, varchar region_id);

Descrição

Carrega uma região definida de dados tiger em uma Topologia PostGIS. As faces, nós e limites são transformados em um sistema de referência espacial de uma topologia alvo e pontos são estalados à tolerância da topologia alvo. As faces, nós e limites criados, mantêm as mesmas identidades dos originais dos dados Tiger, para os datasets serem reconciliados mais facilmente no futuro com os dados tiger. Retorna detalhes resumidos do processo.

Seria útil, por exemplo, para dividir em novos distritos os dados onde você requer que os polígonos formados sigam as linhas centrais das ruas e para os polígonos resultantes não se sobreporem.

Note
Esta função confia nos dados tiger, bem como o módulo de instalação da topologia do PostGIS. para maiores informações, veja: Chapter 8 e Section 2.2.3. Se você não carregou os dados cobrindo a região de interesse, nenhum relato de topologia será criado, Esta função também falhará se você não tiver criado uma topologia usando as funções dela.

Note
A maior parte dos erros de validação de topologia são resultados de issues de tolerância, nas quais depois da transformação os pontos limites não se alinham ou sobrepõem. Para remediar esta situação, você talvez queira aumentar ou diminuir a precisão, se você tiver falhas na validação da topologia.

Argumentos obrigatórios:

1. `topo_name` O nome de uma topologia PostGIS para carregar dados.
2. `region_type` O tipo de região limitadora. Atualmente, somente`place` e `county` são suportados. O plano é ter muitas mais. Esta é a table para investigar para definir os limites da região. ex: `tiger.place`, `tiger.county`
3. `region_id` Isto é o que o Tiger chama de geoid. É o único identificador da região na table. Para lugar é a `plcidfp` coluna em `tiger.place`. Para condado é a `cntyidfp` coluna `tiger.county`

Disponibilidade: 2.0.0

Exemplo: Topologia de Boston, Massachusetts

Criar uma topologia para Boston, Massachusetts em Mass State Plane Feet (2249), com tolerância 0.25 feet e então carregar na cidade de Boston faces, limites e nós tiger.

```
SELECT topology.CreateTopology('topo_boston', 2249, 0.25);
createtopology
--------------
   15
-- 60,902 ms ~ 1 minute on windows 7 desktop running 9.1 (with 5 states tiger data loaded)
SELECT tiger.topology_load_tiger('topo_boston', 'place', '2507000');
-- topology_loader_tiger --
29722 edges holding in temporary. 11108 faces added. 1875 edges of faces added.  20576 ↩
    nodes added.
19962 nodes contained in a face.  0 edge start end corrected.  31597 edges added.

-- 41 ms --
SELECT topology.TopologySummary('topo_boston');
 -- topologysummary--
Topology topo_boston (15), SRID 2249, precision 0.25
20576 nodes, 31597 edges, 11109 faces, 0 topogeoms in 0 layers

-- 28,797 ms to validate yeh returned no errors --
SELECT * FROM
    topology.ValidateTopology('topo_boston');

       error       |   id1    |    id2
-------------------+----------+-----------
```

Exemplo: Suffolk, topologia de Massachusetts

Criar uma topologia para uffolk, Massachusetts in Mass State Plane Meters (26986), com tolerância 0.25 metros e então carregar no condado de Suffolk faces, limites e nós tiger.

```
SELECT topology.CreateTopology('topo_suffolk', 26986, 0.25);
-- this took 56,275 ms ~ 1 minute on Windows 7 32-bit with 5 states of tiger loaded
-- must have been warmed up after loading boston
SELECT tiger.topology_load_tiger('topo_suffolk', 'county', '25025');
-- topology_loader_tiger --
 36003 edges holding in temporary. 13518 faces added. 2172 edges of faces added.
 24761 nodes added.  24075 nodes contained in a face.  0 edge start end corrected.  38175  ↩
    edges added.
-- 31 ms --
SELECT topology.TopologySummary('topo_suffolk');
 -- topologysummary--
 Topology topo_suffolk (14), SRID 26986, precision 0.25
24761 nodes, 38175 edges, 13519 faces, 0 topogeoms in 0 layers

-- 33,606 ms to validate --
SELECT * FROM
    topology.ValidateTopology('topo_suffolk');

       error       |   id1    |    id2
-------------------+----------+-----------
 coincident nodes  | 81045651 |  81064553
 edge crosses node | 81045651 |  85737793
 edge crosses node | 81045651 |  85742215
 edge crosses node | 81045651 | 620628939
 edge crosses node | 81064553 |  85697815
 edge crosses node | 81064553 |  85728168
 edge crosses node | 81064553 |  85733413
```

Veja Também

Cria topologia, CreateTopoGeom, TopologySummary, ValidateTopology

11.2.18 Set_Geocode_Setting

Set_Geocode_Setting — Estabelece uma configuração que afeta comportamento das funções geocoder.

Synopsis

text **Set_Geocode_Setting**(text setting_name, text setting_value);

Descrição

Estabelece valor de uma configuração específica armazenada na table `tiger.geocode_settings`. Configurações permitem que você comute depuração de funções. Os planos futuros serão para controlar avaliação com configurações. A lista atual de configurações está em:

Disponibilidade: 2.1.0

Exemplo da configuração de retornar depuração

Se você executar Geocode quando esta função for verdade, o log NOTICE irá gerar horas e pesquisas.

```
SELECT set_geocode_setting('debug_geocode_address', 'true') As result;
result
---------
true
```

Veja Também

Get_Geocode_Setting

Chapter 12

PostGIS Special Functions Index

12.1 PostGIS Aggregate Functions

The functions below are spatial aggregate functions that are used in the same way as SQL aggregate function such as `sum` and `average`.

- ST_3DExtent - Aggregate function that returns the 3D bounding box of geometries.
- ST_3DUnion - Perform 3D union.
- ST_AsFlatGeobuf - Return a FlatGeobuf representation of a set of rows.
- ST_AsGeobuf - Return a Geobuf representation of a set of rows.
- ST_AsMVT - Aggregate function returning a MVT representation of a set of rows.
- ST_ClusterIntersecting - Aggregate function that clusters input geometries into connected sets.
- ST_ClusterWithin - Aggregate function that clusters input geometries by separation distance.
- ST_CoverageUnion - Computes the union of a set of polygons forming a coverage by removing shared edges.
- ST_Extent - Aggregate function that returns the bounding box of geometries.
- ST_GeomCollFromText - Creates a GeometryCollection or Multi* geometry from a set of geometries.
- ST_MakeLine - Cria uma Linestring de ponto, multiponto ou linha das geometrias.
- ST_MemUnion - Aggregate function which unions geometries in a memory-efficent but slower way
- ST_Polygonize - Computes a collection of polygons formed from the linework of a set of geometries.
- ST_SameAlignment - Retorna verdade se os rasters têm a mesma inclinação, escala, referência espacial, e deslocamento (pixeis podem ser colocados na mesma grade sem cortar eles) e falso se eles não notificarem problemas detalhados.
- ST_Union - Computes a geometry representing the point-set union of the input geometries.
- TopoElementArray_Agg - Returns a topoelementarray for a set of element_id, type arrays (topoelements).

12.2 PostGIS Window Functions

The functions below are spatial window functions that are used in the same way as SQL window functions such as `row_number()`, `lead()`, and `lag()`. They must be followed by an `OVER()` clause.

- ST_ClusterDBSCAN - Window function that returns a cluster id for each input geometry using the DBSCAN algorithm.
- ST_ClusterIntersectingWin - Window function that returns a cluster id for each input geometry, clustering input geometries into connected sets.
- ST_ClusterKMeans - Window function that returns a cluster id for each input geometry using the K-means algorithm.
- ST_ClusterWithinWin - Window function that returns a cluster id for each input geometry, clustering using separation distance.
- ST_CoverageInvalidEdges - Window function that finds locations where polygons fail to form a valid coverage.
- ST_CoverageSimplify - Window function that simplifies the edges of a polygonal coverage.

12.3 PostGIS SQL-MM Compliant Functions

The functions given below are PostGIS functions that conform to the SQL/MM 3 standard

- ST_3DArea - Computa a área de geometrias de superfície 3D. Irá retornar 0 para sólidos. Descrição Disponibilidade: 2.1.0 This method needs SFCGAL backend. This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 8.1, 10.5 This function supports 3d and will not drop the z-index. This function supports Polyhedral surfaces. This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).
- ST_3DDWithin - Tests if two 3D geometries are within a given 3D distance Description Returns true if the 3D distance between two geometry values is no larger than distance distance_of_srid. The distance is specified in units defined by the spatial reference system of the geometries. For this function to make sense the source geometries must be in the same coordinate system (have the same SRID). This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries. This function supports 3d and will not drop the z-index. This function supports Polyhedral surfaces. This method implements the SQL/MM specification. SQL-MM ? Availability: 2.0.0
- ST_3DDifference - Representar diferença 3D Descrição Retorna aquela parte de geom1 que não faz parte de geom2. Disponibilidade: 2.2.0 This method needs SFCGAL backend. This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1 This function supports 3d and will not drop the z-index. This function supports Polyhedral surfaces. This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).
- ST_3DDistance - Para tipo geometria, retorna a menor distância cartesiana 3-dimensional (baseado no sistema de referência espacial) entre duas geometrias em unidades projetadas. Descrição Para tipo geometria, retorna a menor distância cartesiana 3-dimensional entre duas geometrias em unidades projetadas (spatial ref units). This function supports 3d and will not drop the z-index. This function supports Polyhedral surfaces. This method implements the SQL/MM specification. SQL-MM ISO/IEC 13249-3 Disponibilidade: 2.0.0 Alterações: 2.2.0 - Em caso de 2D e 3D, o Z não é mais 0 para Z perdido. Changed: 3.0.0 - SFCGAL version removed
- ST_3DIntersection - Representar intersecção 3D Descrição Retorna uma geometria que é dividida entre geom1 e geom2 Disponibilidade: 2.1.0 This method needs SFCGAL backend. This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1 This function supports 3d and will not drop the z-index. This function supports Polyhedral surfaces. This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).
- ST_3DIntersects - Tests if two geometries spatially intersect in 3D - only for points, linestrings, polygons, polyhedral surface (area) Description Overlaps, Touches, Within all imply spatial intersection. If any of the aforementioned returns true, then the geometries also spatially intersect. Disjoint implies false for spatial intersection. This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries. Changed: 3.0.0 SFCGAL backend removed, GEOS backend supports TINs. Availability: 2.0.0 This function supports 3d and will not drop the z-index. This function supports Polyhedral surfaces. This function supports Triangles and Triangulated Irregular Network Surfaces (TIN). This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1

- ST_3DLength - Retorna o centro geométrico de uma geometria. Descrição Retorna o comprimento 3-dimensional ou 2-dimensional da geometria se for uma linestring ou multi-linestring. Para linhas 2-d, ela só retornará o comprimento 2-d (o mesmo da ST_Length e ST_Length2D) This function supports 3d and will not drop the z-index. This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 7.1, 10.3 Alterações: 2.0.0 Nas versões anteriores era chamado de ST_Length3D
- ST_3DPerimeter - Retorna o centro geométrico de uma geometria. Descrição Retorna o perímetro 3-dimensional da geometria, se for um polígono ou multi-polígono. Se a geometria for 2-dimensional, então retorna o perímetro 2-dimensional. This function supports 3d and will not drop the z-index. This method implements the SQL/MM specification. SQL-MM ISO/IEC 13249-3: 8.1, 10.5 Alterações: 2.0.0 Nas versões anteriores era chamado de ST_Perimeter3D
- ST_3DUnion - Perform 3D union. Descrição Disponibilidade: 2.2.0 Availability: 3.3.0 aggregate variant was added This method needs SFCGAL backend. This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1 This function supports 3d and will not drop the z-index. This function supports Polyhedral surfaces. This function supports Triangles and Triangulated Irregular Network Surfaces (TIN). Aggregate variant: returns a geometry that is the 3D union of a rowset of geometries. The ST_3DUnion() function is an "aggregate" function in the terminology of PostgreSQL. That means that it operates on rows of data, in the same way the SUM() and AVG() functions do and like most aggregates, it also ignores NULL geometries.
- ST_AddEdgeModFace - Adiciona um novo limite e, se uma face for dividida, modifica a face original e adiciona uma nova face. Descrição Adiciona um novo limite e, se uma face for dividida, modifica a face original e adiciona uma nova. Se possível, a face nova será criada no lado esquerdo do novo limite. Isto não será possível se a face do lado esquerdo precisar ser a face universal (sem limites). Retorna a id do novo limite adicionado. Atualiza todos os limites existentes e relacionamentos em conformidade. Se algum argumento for nulo, os nós são desconhecidos (devem existir na table node do esquema de topologia), a acurve não é uma LINESTRING, o anode e anothernode não são os pontos de começo e fim da acurve, logo, um erro é lançado. Se o sistema de referência espacial (srid) da geometria acurve não for o mesmo da topologia, uma exceção é lançada. Disponibilidade: 2.0 This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.13
- ST_AddEdgeNewFaces - Adiciona um novo limite e, se uma face for dividida, deleta a face original e substitui por duas novas faces. Descrição Adiciona um novo limite e, se uma face for dividida, deleta a face original e substitui por duas novas faces. Retorna a id do novo limite adicionado. Atualiza todos os limites existentes e relacionamentos em conformidade. Se algum argumento for nulo, os nós são desconhecidos (devem existir na table node do esquema de topologia), a acurve não é uma LINESTRING, o anode e anothernode não são os pontos de começo e fim da acurve, logo, um erro é lançado. Se o sistema de referência espacial (srid) da geometria acurve não for o mesmo da topologia, uma exceção é lançada. Disponibilidade: 2.0 This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.12
- ST_AddIsoEdge - Adiciona um limite isolado definido pela geometria alinestring a uma topologia conectando dois nós isoladosanode e anothernode e retorna a nova id do novo limite. Descrição Adiciona um limite isolado definido pela geometria alinestring a uma topologia conectando dois nós isoladosanode e anothernode e retorna a nova id do novo limite. Se o sistema de referência espacial (srid) da geometria alinestring não for o mesmo da topologia, qualquer argumento de entrada é nulo, ou is nós estão contidos em mais de uma face, ou eles são o começo ou fim de um limite existente, então, uma exceção é aberta. Se a alinestring não está dentro da face da face o anode e anothernode pertence, então, uma exceção é aberta. Se o anode e anothernode não são os pontos de começo e fim da alinestring então, uma exceção é aberta. Availability: 1.1 This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.4
- ST_AddIsoNode - Adiciona um nó isolado a uma face em uma topologia e retorna a id do novo nó. Se a face é nula, o nó continua sendo criado. Descrição Adiciona um nó isolado com a localização do ponto apoint com uma face existente com faceid aface a uma topologia atopology e retorna a nodeid do novo nó. O sistema de referência espacial (srid) da geometria pontual não é o mesmo que a topologia, o apoint não é uma geometria pontual, o ponto é nulo, ou o ponto intersecta um limite existente (mesmo nos limites), então uma exceção é aberta. Se o ponto já existe como um nó, uma exceção é aberta. Se aface não é nula e o apoint não está dentro da face, então, uma exceção é aberta. Availability: 1.1 This method implements the SQL/MM specification. SQL-MM: Topo-Net Routines: X+1.3.1
- ST_Area - Retorna o centro geométrico de uma geometria. Descrição Retorna a área da geometria se for um polígono ou multipolígono. Retorna a medida do comprimento de um valor ST_Surface ou ST_MultiSurface. Para geometria, uma área cartesiana 2D é determinada com unidades especificadas pelo SRID. Para geografia, por padrão, ela é determinada em um esferoide com unidade em metros quadrados. Para medir a esfera mais rápida, mas menos precisa, use: ST_Area(geog,false). Melhorias: 2.0.0 - suporte a superfícies 2D poliédricas foi introduzido. Melhorias: 2.2.0 - medição em esferoides desempenhada com GeographicLib para uma melhor precisão e força. Requer Proj >= 4.9.0 para tirar vantagem da nova característica.

Changed: 3.0.0 - does not depend on SFCGAL anymore. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. This method implements the SQL/MM specification. SQL-MM 3: 8.1.2, 9.5.3 This function supports Polyhedral surfaces. Para superfícies poliédricas, somente suporta superfícies poliédricas 2D (não 2.5D). Para 2.5D, pode ser dada uma resposta não zero, mas somente para as faces que se encaixam completamente no plano XY.

- ST_AsBinary - Return the OGC/ISO Well-Known Binary (WKB) representation of the geometry/geography without SRID meta data. Descrição Returns the OGC/ISO Well-Known Binary (WKB) representation of the geometry. The first function variant defaults to encoding using server machine endian. The second function variant takes a text argument specifying the endian encoding, either little-endian ('NDR') or big-endian ('XDR'). WKB format is useful to read geometry data from the database and maintaining full numeric precision. This avoids the precision rounding that can happen with text formats such as WKT. To perform the inverse conversion of WKB to PostGIS geometry use . The OGC/ISO WKB format does not include the SRID. To get the EWKB format which does include the SRID use The default behavior in PostgreSQL 9.0 has been changed to output bytea in hex encoding. If your GUI tools require the old behavior, then SET bytea_output='escape' in your database. Melhorias: 2.0.0 suporte para superfícies poliédricas, triângulos e TINs introduzido. Melhorias: 2.0.0 suporte para maiores dimensões de coordenadas foi introduzido. Melhorias: 2.0.0 suporte para edian especificando com geografia foi introduzido. Disponibilidade: 1.5.0 suporte para geografia foi introduzido. Alterações: 2.0.0 Entrada para esta função não pode ser desconhecida -- deve ser geometria. Construções como ST_AsBinary('POINT(1 2)') não são mais válidas e você terá n st_asbinary(desconhecido) não é um erro único. Códigos assim, precisam ser alterados para ST_AsBinary('POINT(1 2)'::geometry);. Se não for possível, instale: legacy.sql. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.1 This method implements the SQL/MM specification. SQL-MM 3: 5.1.37 This method supports Circular Strings and Curves. This function supports Polyhedral surfaces. This function supports Triangles and Triangulated Irregular Network Surfaces (TIN). This function supports 3d and will not drop the z-index.

- ST_AsGML - Retorna a geometria como uma versão GML com 2 ou 3 elementos. Descrição Return the geometry as a Geography Markup Language (GML) element. The version parameter, if specified, may be either 2 or 3. If no version parameter is specified then the default is assumed to be 2. The maxdecimaldigits argument may be used to reduce the maximum number of decimal places used in output (defaults to 15). Using the maxdecimaldigits parameter can cause output geometry to become invalid. To avoid this use with a suitable gridsize first. GML 2 refere-se a versão 2.1.2 , GML 3 para a versão 3.1.1 O argumento "opções" é um bitfield. Ele poderia ser usado para definir o tipo de saída CRS na saída GML, e para declarar dados como lat/lon: 0: GML Short CRS (ex: EPSG:4326), valor padrão 1: GML Long CRS (ex: urn:ogc:def:crs:EPSG::4326) 2: Para GML 3 somente, remove srsDimension atribuída da saída. 4: Para GML 3 somente, use <LineString> em vez de <Curve> tag para linhas. 16: Declara que dados são lat/lon (ex: srid=4326). O padrão é supor que os dados são planos. Esta opção é útil apenas para saída GML 3.1.1, relacionada a ordem do eixo. Então, se você configurá-la, ela irá trocar as coordenadas, deixando a ordem sendo lat lon em vez do banco de dados. 32: Gera a caixa da geometria (envelope). O argumento 'namespace prefix' pode ser usado para especificar um namespace prefix personalizado ou nenhum prefixo (se vazio). Se nulo ou omitido, o prefixo 'gml' é usado Disponibilidade: 1.3.2 Disponibilidade: 1.5.0 suporte para geografia foi introduzido. Melhorias: 2.0.0 prefixo suportado foi introduzido. A opção 4 para o GML3 foi introduzida para permitir a utilização da LineString em vez da tag Curva para linhas. O suporte GML3 para superfícies poliédricas e TINS foi introduzidos. A Opção 32 foi introduzida para gerar a caixa. Alterações: 2.0.0 use argumentos nomeados por padrão Melhorias: 2.1.0 suporte para id foi introduzido, para GML 3. Somente a versão 3+ de ST_AsGML suporta superfícies poliédricas e TINS. This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 17.2 This function supports 3d and will not drop the z-index. This function supports Polyhedral surfaces. This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

- ST_AsText - Retorna a representação de texto bem conhecida (WKT) da geometria/geografia sem os meta dados do SRID. Descrição Returns the OGC Well-Known Text (WKT) representation of the geometry/geography. The optional maxdecimaldigits argument may be used to limit the number of digits after the decimal point in output ordinates (defaults to 15). To perform the inverse conversion of WKT representation to PostGIS geometry use . The standard OGC WKT representation does not include the SRID. To include the SRID as part of the output representation, use the non-standard PostGIS function The textual representation of numbers in WKT may not maintain full floating-point precision. To ensure full accuracy for data storage or transport it is best to use Well-Known Binary (WKB) format (see and maxdecimaldigits). Using the maxdecimaldigits parameter can cause output geometry to become invalid. To avoid this use with a suitable gridsize first. Disponibilidade: 1.5 - suporte para geografia foi introduzido. Enhanced: 2.5 - optional parameter precision introduced. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.1 This method implements the SQL/MM specification. SQL-MM 3: 5.1.25 This method supports Circular Strings and Curves.

- ST_Boundary - Retorna o encerramento da borda combinatória dessa geometria. Descrição Retorna o encerramento do limite combinatório dessa geometria. O limite combinatório é definido com descrito na seção 3.12.3.2 do OGC SPEC. Porque o resultado dessa função é um encerramento, e por isso topologicamente fechado, o limite resultante pode ser representado

usando geometrias primitivas representacionais como foi discutido no OGC SPEC, seção 3.12.2. Desempenhado pelo módulo GEOS Anterior a 2.0.0, essa função abre uma exceção se usada com GEOMETRYCOLLECTION. A partir do 2.0.0 ela vai retornar NULA (entrada não suportada). This method implements the OGC Simple Features Implementation Specification for SQL 1.1. OGC SPEC s2.1.1.1 This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1.17 This function supports 3d and will not drop the z-index. Melhorias: 2.1.0 suporte para Triângulo foi introduzido Changed: 3.2.0 support for TIN, does not use geos, does not linearize curves

- ST_Buffer - Computes a geometry covering all points within a given distance from a geometry. Descrição Computes a POLYGON or MULTIPOLYGON that represents all points whose distance from a geometry/geography is less than or equal to a given distance. A negative distance shrinks the geometry rather than expanding it. A negative distance may shrink a polygon completely, in which case POLYGON EMPTY is returned. For points and lines negative distances always return empty results. For geometry, the distance is specified in the units of the Spatial Reference System of the geometry. For geography, the distance is specified in meters. The optional third parameter controls the buffer accuracy and style. The accuracy of circular arcs in the buffer is specified as the number of line segments used to approximate a quarter circle (default is 8). The buffer style can be specifed by providing a list of blank-separated key=value pairs as follows: 'quad_segs=#' : number of line segments used to approximate a quarter circle (default is 8). 'endcap=round|flat|square' : endcap style (defaults to "round"). 'butt' is accepted as a synonym for 'flat'. 'join=round|mitre|bevel' : join style (defaults to "round"). 'miter' is accepted as a synonym for 'mitre'. 'mitre_limit=#.#' : mitre ratio limit (only affects mitered join style). 'miter_limit' is accepted as a synonym for 'mitre_limit'. 'side=both|left|right' : 'left' or 'right' performs a single-sided buffer on the geometry, with the buffered side relative to the direction of the line. This is only applicable to LINESTRING geometry and does not affect POINT or POLYGON geometries. By default end caps are square. For geography this is a thin wrapper around the geometry implementation. It determines a planar spatial reference system that best fits the bounding box of the geography object (trying UTM, Lambert Azimuthal Equal Area (LAEA) North/South pole, and finally Mercator). The buffer is computed in the planar space, and then transformed back to WGS84. This may not produce the desired behavior if the input object is much larger than a UTM zone or crosses the dateline Buffer output is always a valid polygonal geometry. Buffer can handle invalid inputs, so buffering by distance 0 is sometimes used as a way of repairing invalid polygons. can also be used for this purpose. Buffering is sometimes used to perform a within-distance search. For this use case it is more efficient to use . This function ignores the Z dimension. It always gives a 2D result even when used on a 3D geometry. Enhanced: 2.5.0 - ST_Buffer geometry support was enhanced to allow for side buffering specification side=both|left|right. Availability: 1.5 - ST_Buffer was enhanced to support different endcaps and join types. These are useful for example to convert road linestrings into polygon roads with flat or square edges instead of rounded edges. Thin wrapper for geography was added. Desempenhado pelo módulo GEOS. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.3 This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1.30

- ST_Centroid - Retorna o centro geométrico de uma geometria. Descrição Computes a point which is the geometric center of mass of a geometry. For [MULTI]POINTs, the centroid is the arithmetic mean of the input coordinates. For [MULTI]LINESTRINGs, the centroid is computed using the weighted length of each line segment. For [MULTI]POLYGONs, the centroid is computed in terms of area. If an empty geometry is supplied, an empty GEOMETRYCOLLECTION is returned. If NULL is supplied, NULL is returned. If CIRCULARSTRING or COMPOUNDCURVE are supplied, they are converted to linestring with CurveToLine first, then same than for LINESTRING For mixed-dimension input, the result is equal to the centroid of the component Geometries of highest dimension (since the lower-dimension geometries contribute zero "weight" to the centroid). Note that for polygonal geometries the centroid does not necessarily lie in the interior of the polygon. For example, see the diagram below of the centroid of a C-shaped polygon. To construct a point guaranteed to lie in the interior of a polygon use . New in 2.3.0 : supports CIRCULARSTRING and COMPOUNDCURVE (using CurveToLine) Availability: 2.4.0 support for geography was introduced. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. This method implements the SQL/MM specification. SQL-MM 3: 8.1.4, 9.5.5

- ST_ChangeEdgeGeom - Modifica a forma de um limite sem afetar a estrutura da topologia. Descrição Modifica a forma de um limite sem afetar a estrutura da topologia. If any arguments are null, the given edge does not exist in the edge table of the topology schema, the acurve is not a LINESTRING, or the modification would change the underlying topology then an error is thrown. Se o sistema de referência espacial (srid) da geometria acurve não for o mesmo da topologia, uma exceção é lançada. Se a nova acurve não for simples, um erro é lançado. Se mover o limite de uma posição antiga acertar um obstáculo, um erro é lançado. Disponibilidade: 1.1.0 Melhorias: 2.0.0 adiciona execução da consistência topológica This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details X.3.6

- ST_Contains - Tests if every point of B lies in A, and their interiors have a point in common Description Returns TRUE if geometry A contains geometry B. A contains B if and only if all points of B lie inside (i.e. in the interior or boundary of) A (or equivalently, no points of B lie in the exterior of A), and the interiors of A and B have at least one point in common.

In mathematical terms: ST_Contains(A, B) ⇔ (A ⋂ B = B) ∧ (Int(A) ⋂ Int(B) ≠ ∅) The contains relationship is reflexive: every geometry contains itself. (In contrast, in the predicate a geometry does not properly contain itself.) The relationship is antisymmetric: if ST_Contains(A,B) = true and ST_Contains(B,A) = true, then the two geometries must be topologically equal (ST_Equals(A,B) = true). ST_Contains is the converse of . So, ST_Contains(A,B) = ST_Within(B,A). Because the interiors must have a common point, a subtlety of the definition is that polygons and lines do not contain lines and points lying fully in their boundary. For further details see Subtleties of OGC Covers, Contains, Within. The predicate provides a more inclusive relationship. This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries. To avoid index use, use the function _ST_Contains. Performed by the GEOS module Enhanced: 2.3.0 Enhancement to PIP short-circuit extended to support MultiPoints with few points. Prior versions only supported point in polygon. Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION Do not use this function with invalid geometries. You will get unexpected results. NOTE: this is the "allowable" version that returns a boolean, not an integer. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.2 // s2.1.13.3 - same as within(geometry B, geometry A) This method implements the SQL/MM specification. SQL-MM 3: 5.1.31

- ST_ConvexHull - Computes the convex hull of a geometry. Descrição Computes the convex hull of a geometry. The convex hull is the smallest convex geometry that encloses all geometries in the input. One can think of the convex hull as the geometry obtained by wrapping an rubber band around a set of geometries. This is different from a concave hull which is analogous to "shrink-wrapping" the geometries. A convex hull is often used to determine an affected area based on a set of point observations. In the general case the convex hull is a Polygon. The convex hull of two or more collinear points is a two-point LineString. The convex hull of one or more identical points is a Point. This is not an aggregate function. To compute the convex hull of a set of geometries, use to aggregate them into a geometry collection (e.g. ST_ConvexHull(ST_Collect(geom)). Desempenhado pelo módulo GEOS This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.3 This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1.16 This function supports 3d and will not drop the z-index.

- ST_CoordDim - Retorna a dimensão da coordenada do valor ST_Geometry. Descrição Retorna a dimensão da coordenada do valor ST_Geometry. Esse é o pseudônimo condescendente do MM para This method implements the OGC Simple Features Implementation Specification for SQL 1.1. This method implements the SQL/MM specification. SQL-MM 3: 5.1.3 This method supports Circular Strings and Curves. This function supports 3d and will not drop the z-index. This function supports Polyhedral surfaces. This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

- ST_CreateTopoGeo - Adiciona uma coleção de geometrias para uma dada topologia vazia e retorna uma mensagem detalhando sucesso. Descrição Adiciona uma coleção de geometrias para uma dada topologia vazia e retorna uma mensagem detalhando sucesso. Útil para popular uma topologia vazia. Disponibilidade: 2.0 This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details -- X.3.18

- ST_Crosses - Tests if two geometries have some, but not all, interior points in common Description Compares two geometry objects and returns true if their intersection "spatially crosses"; that is, the geometries have some, but not all interior points in common. The intersection of the interiors of the geometries must be non-empty and must have dimension less than the maximum dimension of the two input geometries, and the intersection of the two geometries must not equal either geometry. Otherwise, it returns false. The crosses relation is symmetric and irreflexive. In mathematical terms: ST_Crosses(A, B) ⇔ (dim(Int(A) ⋂ Int(B)) < max(dim(Int(A)), dim(Int(B)))) ∧ (A ⋂ B ≠ A) ∧ (A ⋂ B ≠ B) Geometries cross if their DE-9IM Intersection Matrix matches: T*T****** for Point/Line, Point/Area, and Line/Area situations T*****T** for Line/Point, Area/Point, and Area/Line situations 0******** for Line/Line situations the result is false for Point/Point and Area/Area situations The OpenGIS Simple Features Specification defines this predicate only for Point/Line, Point/Area, Line/Line, and Line/Area situations. JTS / GEOS extends the definition to apply to Line/Point, Area/Point and Area/Line situations as well. This makes the relation symmetric. This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries. Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.13.3 This method implements the SQL/MM specification. SQL-MM 3: 5.1.29

- ST_CurveToLine - Converts a geometry containing curves to a linear geometry. Descrição Converts a CIRCULAR STRING to regular LINESTRING or CURVEPOLYGON to POLYGON or MULTISURFACE to MULTIPOLYGON. Useful for outputting to devices that can't support CIRCULARSTRING geometry types Converts a given geometry to a linear geometry. Each curved geometry or segment is converted into a linear approximation using the given `tolerance` and options (32 segments per quadrant and no options by default). The 'tolerance_type' argument determines interpretation of the `tolerance` argument. It can take the following values: 0 (default): Tolerance is max segments per quadrant. 1: Tolerance is max-deviation of line from curve, in source units. 2: Tolerance is max-angle, in radians, between generating radii. The 'flags' argument is a bitfield. 0 by default. Supported bits are: 1: Symmetric (orientation idependent) output. 2: Retain angle, avoids reducing angles (segment lengths)

when producing symmetric output. Has no effect when Symmetric flag is off. Availability: 1.3.0 Enhanced: 2.4.0 added support for max-deviation and max-angle tolerance, and for symmetric output. Enhanced: 3.0.0 implemented a minimum number of segments per linearized arc to prevent topological collapse. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. This method implements the SQL/MM specification. SQL-MM 3: 7.1.7 This function supports 3d and will not drop the z-index. This method supports Circular Strings and Curves.

- ST_Difference - Computes a geometry representing the part of geometry A that does not intersect geometry B. Description Returns a geometry representing the part of geometry A that does not intersect geometry B. This is equivalent to A - ST_Intersection(A,B). If A is completely contained in B then an empty atomic geometry of appropriate type is returned. This is the only overlay function where input order matters. ST_Difference(A, B) always returns a portion of A. If the optional gridSize argument is provided, the inputs are snapped to a grid of the given size, and the result vertices are computed on that same grid. (Requires GEOS-3.9.0 or higher) Performed by the GEOS module Enhanced: 3.1.0 accept a gridSize parameter. Requires GEOS >= 3.9.0 to use the gridSize parameter. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.3 This method implements the SQL/MM specification. SQL-MM 3: 5.1.20 This function supports 3d and will not drop the z-index. However, the result is computed using XY only. The result Z values are copied, averaged or interpolated.

- ST_Dimension - Retorna a dimensão da coordenada do valor ST_Geometry. Descrição A dimensão herdada desse objeto geométrico, que deve ser menor que ou igual à dimensão coordenada. OGC SPEC s2.1.1.1 - retorna 0 para PONTO, 1 para LINESTRING, 2 para POLÍGONO, e a dimensão mais larga dos componentes de uma COLEÇÃODEGEOMETRIA. Se desconhecida (geometria vazia) nula é retornada. This method implements the SQL/MM specification. SQL-MM 3: 5.1.2 Melhorias: 2.0.0 suporte para superfícies poliédricas e TINs foi introduzido. Não abre mais exceção se uma geometria vazia é dada. Anterior à 2.0.0, essa função abre uma exceção se usada com uma geometria vazia. This function supports Polyhedral surfaces. This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

- ST_Disjoint - Tests if two geometries have no points in common Description Returns true if two geometries are disjoint. Geometries are disjoint if they have no point in common. If any other spatial relationship is true for a pair of geometries, they are not disjoint. Disjoint implies that is false. In mathematical terms: ST_Disjoint(A, B) $\Leftrightarrow$ A $\bigcap$ B = $\varnothing$ Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION Performed by the GEOS module This function call does not use indexes. A negated predicate can be used as a more performant alternative that uses indexes: ST_Disjoint(A,B) = NOT ST_Intersects(A,B) NOTE: this is the "allowable" version that returns a boolean, not an integer. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.2 //s2.1.13.3 - a.Relate(b, 'FF*FF****') This method implements the SQL/MM specification. SQL-MM 3: 5.1.26

- ST_Distance - Retorna a linha 3-dimensional mais longa entre duas geometrias Descrição Para tipo geometria, retorna a menor distância cartesiana 3-dimensional entre duas geometrias em unidades projetadas (spatial ref units). For types defaults to return the minimum geodesic distance between two geographies in meters, compute on the spheroid determined by the SRID. If use_spheroid is false, a faster spherical calculation is used. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. This method implements the SQL/MM specification. SQL-MM 3: 5.1.23 This method supports Circular Strings and Curves. Disponibilidade: 1.5.0 suporte de geografia foi introduzido em 1.5. Melhorias na velocidade para planar para lidar melhor com mais ou maiores vértices de geometrias. Melhorias: 2.1.0 velocidade melhorada para geografia. Veja Making Geography faster para mais detalhes. Melhorias: 2.1.0 - suporte para geometrias curvas foi introduzido. Melhorias: 2.2.0 - medição em esferoides desempenhada com GeographicLib para uma melhor precisão e força. Requer Proj >= 4.9.0 para tirar vantagem da nova característica. Changed: 3.0.0 - does not depend on SFCGAL anymore.

- ST_EndPoint - Retorna o número de pontos em um valor ST_LineString ou ST_CircularString. Descrição Retorna ao último ponto de uma LINESTRING ou CIRCULARLINESTRING geometria como um PONTO ou NULO se o parâmetro de entrada não é uma LINESTRING. This method implements the SQL/MM specification. SQL-MM 3: 7.1.4 This function supports 3d and will not drop the z-index. This method supports Circular Strings and Curves. Alterações: 2.0.0 não funciona mais com geometrias de multilinestrings. Em verões mais antigas do PostGIS -- uma linha multilinestring sozinha trabalharia normalmente com essa função e voltaria o ponto de início. Na 2.0.0 ela retorna NULA como qualquer outra multilinestring. O antigo comportamento não foi uma característica documentada, mas as pessoas que consideravam que tinham seus dados armazenados como uma LINESTRING, agora podem experimentar essas que retornam NULAS em 2.0.

- ST_Envelope - Retorna uma geometria representando a precisão da dobrada (float8) da caixa limitada da geometria fornecida. Descrição Retorna o limite mínimo da caixa float8 para a geometria fornecida, com uma geometria. O polígono é definido pelos pontos de canto da caixa limitada ((MINX, MINY), (MINX, MAXY), (MAXX, MAXY), (MAXX, MINY), (MINX, MINY)). (PostGIS irá adicionar uma ZMIN/ZMAX coordenada também). Casos degenerados (linhas verticais, pontos) irão retornar como uma geometria de dimensão menor que POLÍGONO, ie. PONTO ou LINESTRING. Disponibilidade: 1.5.0

comportamento alterado para saída de precisão dupla ao invés de float4 This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.1 This method implements the SQL/MM specification. SQL-MM 3: 5.1.19

- ST_Equals - Tests if two geometries include the same set of points Description Returns true if the given geometries are "topologically equal". Use this for a 'better' answer than '='. Topological equality means that the geometries have the same dimension, and their point-sets occupy the same space. This means that the order of vertices may be different in topologically equal geometries. To verify the order of points is consistent use (it must be noted ST_OrderingEquals is a little more stringent than simply verifying order of points are the same). In mathematical terms: ST_Equals(A, B) ⇔ A = B The following relation holds: ST_Equals(A, B) ⇔ ST_Within(A,B) ∧ ST_Within(B,A) Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.2 This method implements the SQL/MM specification. SQL-MM 3: 5.1.24 Changed: 2.2.0 Returns true even for invalid geometries if they are binary equal

- ST_ExteriorRing - Retorna o número de anéis interiores de um polígono. Descrição Retorna uma line string representando o anel exterior da geometria POLÍGONO. Retorna NULA se a geometria não for um polígono. Isso não funcionará para MULTIPOLÍGONOS. Use em conjunção com ST_Dump para MULTIPOLÍGONOS. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. 2.1.5.1 This method implements the SQL/MM specification. SQL-MM 3: 8.2.3, 8.3.3 This function supports 3d and will not drop the z-index.

- ST_GMLToSQL - Retorna um valor ST_Geometry específico da representação GML. Esse é um heterônimo para ST_GeomFromGML Descrição This method implements the SQL/MM specification. SQL-MM 3: 5.1.50 (exceto para curvas suporte). Disponibilidade:1.5, requer libxml2 1.6+ Melhorias: 2.0.0 suporte para superfícies poliédricas e TIN foi introduzido. Melhorias: 2.0.0 parâmetro opcional padrão srid adicionado.

- ST_GeomCollFromText - Makes a collection Geometry from collection WKT with the given SRID. If SRID is not given, it defaults to 0. Descrição Makes a collection Geometry from the Well-Known-Text (WKT) representation with the given SRID. If SRID is not given, it defaults to 0. OGC SPEC 3.2.6.2 - opção SRID é de uma suíte de conformação Retorna nula se a WKT não for uma GEOMETRYCOLLECTION se você não tem total certeza de que todas suas geometrias WKT são coleções, não use essa função. Ela é mais devagar que a ST_GeomFromText, já que adiciona um passo de validação adicional. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2 This method implements the SQL/MM specification.

- ST_GeomFromText - Retorna um valor ST_Geometry específico da representação de texto bem conhecida (WKT). Descrição Constrói um objeto PostGIS ST_Geometry de uma representação de texto bem conhecida OGC. Existem duas variantes da função ST_GeomFromText. A primeira não pega nenhuma SRID e retorna uma geometria com um sistema de referência espacial indefinido (SRID=0). A segunda pega uma SRID como o segundo argumento e retorna uma geometria que inclui essa SRID como parte dos seus metadados. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2 - opção SRID é da suíte de conformidade. This method implements the SQL/MM specification. SQL-MM 3: 5.1.40 This method supports Circular Strings and Curves. While not OGC-compliant, is faster than ST_GeomFromText and ST_PointFromText. It is also easier to use for numeric coordinate values. is another option similar in speed to and is OGC-compliant, but doesn't support anything but 2D points. Alterações: 2.0.0 Nas primeiras versões do PostGIS, ST_GeomFromText('GEOMETRYCOLLECTION(EMPTY)') foi permitida. Ela agora é ilegal no PostGIS 2.0.0 para melhor se adequar aos padrões SQL/MM. Ela deverá se escrita como ST_GeomFromText('GEOMETRYCOLLECTION EMPTY')

- ST_GeomFromWKB - Criar uma geometria exemplo de um representação bem conhecida de geometria binária (WKB) e SRID opcional. Descrição A função ST_GeomFromWKB, pega uma representação binária bem conhecida de uma geometria e um sistema de referência espacial ID (SRID) e cria um exemplo do tipo apropriado de geometria. Essa função cumpre o papel da Fábrica de Geometria na SQL. Isso é um nome alternativo para ST_WKBToSQL. Se o SRID não for especificado, leva a 0 (desconhecido). This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s3.2.7.2 - o SRID opcional é da suíte de conformidade. This method implements the SQL/MM specification. SQL-MM 3: 5.1.41 This method supports Circular Strings and Curves.

- ST_GeometryFromText - Retorna um valor ST_Geometry especifico da representação de texto estendida bem conhecida (EWKT). Isso é um heterônimo para ST_GeomFromText Descrição This method implements the OGC Simple Features Implementation Specification for SQL 1.1. This method implements the SQL/MM specification. SQL-MM 3: 5.1.40

- ST_GeometryN - Retorna o tipo de geometria de valor ST_Geometry. Descrição Retorna a geometria de 1-base Nth se a geometria é uma GEOMETRYCOLLECTION, (MULTI)POINT, (MULTI)LINESTRING, MULTICURVE ou (MULTI)POLYGON, POLYHEDRALSURFACE. Senão, retorna NULA. O Index é 1-base como para OGC specs desde a versão 0.8.0. Versões

anteriores implementaram isso como 0-base. Se você quiser extrair todas as geometrias, de uma geometria, ST_Dump é mais eficiente e também funcionará para geometrias singulares. Melhorias: 2.0.0 suporte para superfícies poliédricas, triângulos e TIN introduzido. Alterações: 2.0.0. Versões anteriores voltariam NULAS para geometrias únicas. Isso foi alterado para volrtar a geometria para o caso ST_GeometryN(..,1). This method implements the OGC Simple Features Implementation Specification for SQL 1.1. This method implements the SQL/MM specification. SQL-MM 3: 9.1.5 This function supports 3d and will not drop the z-index. This method supports Circular Strings and Curves. This function supports Polyhedral surfaces. This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

- ST_GeometryType - Retorna o tipo de geometria de valor ST_Geometry. Descrição Retorna o tipo da geometria como uma string. EX: 'ST_LineString', 'ST_Polygon','ST_MultiPolygon' etc. Essa função difere de GeometryType(geometria) no caso da string e ST na frente que é retornada, bem como o fato que isso não indicará se a geometria é medida. Melhorias: 2.0.0 suporte a superfícies poliédricas foi introduzido. This method implements the SQL/MM specification. SQL-MM 3: 5.1.4 This function supports 3d and will not drop the z-index. This function supports Polyhedral surfaces.

- ST_GetFaceEdges - Retorna um conjunto de limites ordenados que amarram aface. Descrição Retorna um conjunto de limites ordenados que amarram aface. Cada saída consiste em uma sequência e uma limiteid. Os números das sequências começam com o valor 1. A enumeração dos limites de cada anel começa do limite com o menos identificador. A ordem de limites segue uma regra da mão esquerda (a face amarrada está a esquerda de cada limite direto). Disponibilidade: 2.0 This method implements the SQL/MM specification. SQL-MM 3 Topo-Geo and Topo-Net 3: Routine Details: X.3.5

- ST_GetFaceGeometry - Retorna o polígono na topologia dada com a id de face especificada. Descrição Retorna o polígono na topologia dada com a id de face especificada. Constrói o polígono dos limites fazendo a face. Availability: 1.1 This method implements the SQL/MM specification. SQL-MM 3 Topo-Geo and Topo-Net 3: Routine Details: X.3.16

- ST_InitTopoGeo - Creates a new topology schema and registers it in the topology.topology table. Descrição This is the SQL-MM equivalent of . It lacks options for spatial reference system and tolerance. it returns a text description of the topology creation, instead of the topology id. Availability: 1.1 This method implements the SQL/MM specification. SQL-MM 3 Topo-Geo and Topo-Net 3: Routine Details: X.3.17

- ST_InteriorRingN - Retorna o número de anéis interiores de um polígono. Descrição Retorna o anel linestring Nth interior do polígono. Retorna NULO se a geometria não for um polígono ou o dado N está fora da extensão. Isso não funcionará para MULTIPOLÍGONOS. Use em conjunção com ST_Dump para MULTIPOLÍGONOS. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. This method implements the SQL/MM specification. SQL-MM 3: 8.2.6, 8.3.5 This function supports 3d and will not drop the z-index.

- ST_Intersection - Computes a geometry representing the shared portion of geometries A and B. Description Returns a geometry representing the point-set intersection of two geometries. In other words, that portion of geometry A and geometry B that is shared between the two geometries. If the geometries have no points in common (i.e. are disjoint) then an empty atomic geometry of appropriate type is returned. If the optional gridSize argument is provided, the inputs are snapped to a grid of the given size, and the result vertices are computed on that same grid. (Requires GEOS-3.9.0 or higher) ST_Intersection in conjunction with is useful for clipping geometries such as in bounding box, buffer, or region queries where you only require the portion of a geometry that is inside a country or region of interest. For geography this is a thin wrapper around the geometry implementation. It first determines the best SRID that fits the bounding box of the 2 geography objects (if geography objects are within one half zone UTM but not same UTM will pick one of those) (favoring UTM or Lambert Azimuthal Equal Area (LAEA) north/south pole, and falling back on mercator in worst case scenario) and then intersection in that best fit planar spatial ref and retransforms back to WGS84 geography. This function will drop the M coordinate values if present. If working with 3D geometries, you may want to use SFGCAL based which does a proper 3D intersection for 3D geometries. Although this function works with Z-coordinate, it does an averaging of Z-Coordinate. Performed by the GEOS module Enhanced: 3.1.0 accept a gridSize parameter Requires GEOS >= 3.9.0 to use the gridSize parameter Changed: 3.0.0 does not depend on SFCGAL. Availability: 1.5 support for geography data type was introduced. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.3 This method implements the SQL/MM specification. SQL-MM 3: 5.1.18 This function supports 3d and will not drop the z-index. However, the result is computed using XY only. The result Z values are copied, averaged or interpolated.

- ST_Intersects - Tests if two geometries intersect (they have at least one point in common) Description Returns true if two geometries intersect. Geometries intersect if they have any point in common. For geography, a distance tolerance of 0.00001 meters is used (so points that are very close are considered to intersect). In mathematical terms: ST_Intersects(A, B) $\Leftrightarrow$ A $\bigcap$ B $\neq \varnothing$ Geometries intersect if their DE-9IM Intersection Matrix matches one of: T******** *T******* ***T***** ****T**** Spatial intersection is implied by all the other spatial relationship tests, except , which tests that geometries do NOT intersect.

This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries. Changed: 3.0.0 SFCGAL version removed and native support for 2D TINS added. Enhanced: 2.5.0 Supports GEOMETRYCOLLECTION. Enhanced: 2.3.0 Enhancement to PIP short-circuit extended to support MultiPoints with few points. Prior versions only supported point in polygon. Performed by the GEOS module (for geometry), geography is native Availability: 1.5 support for geography was introduced. For geography, this function has a distance tolerance of about 0.00001 meters and uses the sphere rather than spheroid calculation. NOTE: this is the "allowable" version that returns a boolean, not an integer. This method implements the OGC 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)) This method implements the SQL/MM specification. SQL-MM 3: 5.1.27 This method supports Circular Strings and Curves. This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

- ST_IsClosed - Retorna VERDADEIRO se os pontos de começo e fim da LINESTRING são coincidentes. Para superfície poliédrica está fechada (volumétrica). Descrição Retorna VERDADEIRO se os pontos de começo e fim da LINESTRING são coincidentes. Para superfícies poliédricas, isso lhe diz se a superfície é territorial (aberta) ou volumétrica (fechada). This method implements the OGC Simple Features Implementation Specification for SQL 1.1. This method implements the SQL/MM specification. SQL-MM 3: 7.1.5, 9.3.3 SQL-MM define o resultado do ST_IsClosed(NULO) para ser 0, enquanto o PostGIS retorna NULO. This function supports 3d and will not drop the z-index. This method supports Circular Strings and Curves. Melhorias: 2.0.0 suporte a superfícies poliédricas foi introduzido. This function supports Polyhedral surfaces.

- ST_IsEmpty - Tests if a geometry is empty. Descrição Retorna verdadeiro se essa geometria se é vazia. Se verdadeira, ela representa uma coleção vazia, polígono, ponto etc. SQL-MM define o resultado da ST_IsEmpty(NULA) para ser 0, enquanto o PostGIS retorna NULO. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.1 This method implements the SQL/MM specification. SQL-MM 3: 5.1.7 This method supports Circular Strings and Curves. Alterações: 2.0.0 Nas versões anteriores do PostGIS ST_GeomFromText('GEOMETRYCOLLECTION(EMPTY)') era permitido. Agora isso é ilegal no PostGIS 2.0.0 para se adequar aos padrões SQL/MM.

- ST_IsRing - Tests if a LineString is closed and simple. Descrição Retorna VERDADEIRO se essa LINESTRING for (ST_StartPoint(g) ~= ST_Endpoint(g)) e (não cruzar consigo mesma). This method implements the OGC Simple Features Implementation Specification for SQL 1.1. 2.1.5.1 This method implements the SQL/MM specification. SQL-MM 3: 7.1.6 SQL-MM define o resultado do ST_IsRing(NULO) para ser 0, enquanto o PostGIS retorna NULO.

- ST_IsSimple - Retorna (VERDADEIRA) se essa geometria não tem nenhum ponto irregular, como auto intersecção ou tangenciação. Descrição Retorna verdadeira se essa geometria não tem nenhum ponto geométrico irregular, como auto intersecção ou tangenciação. Para maiores informações na definição OGC da simplicidade e validade das geometrias, use "Ensuring OpenGIS compliancy of geometries" SQL-MM define o resultado da ST_IsSimple(NULA) para ser 0, enquanto o PostGIS retorna NULO. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.1 This method implements the SQL/MM specification. SQL-MM 3: 5.1.8 This function supports 3d and will not drop the z-index.

- ST_IsValid - Tests if a geometry is well-formed in 2D. Description Tests if an ST_Geometry value is well-formed and valid in 2D according to the OGC rules. For geometries with 3 and 4 dimensions, the validity is still only tested in 2 dimensions. For geometries that are invalid, a PostgreSQL NOTICE is emitted providing details of why it is not valid. For the version with the flags parameter, supported values are documented in This version does not print a NOTICE explaining invalidity. For more information on the definition of geometry validity, refer to SQL-MM defines the result of ST_IsValid(NULL) to be 0, while PostGIS returns NULL. Performed by the GEOS module. The version accepting flags is available starting with 2.0.0. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. This method implements the SQL/MM specification. SQL-MM 3: 5.1.9 Neither OGC-SFS nor SQL-MM specifications include a flag argument for ST_IsValid. The flag is a PostGIS extension.

- ST_Length - Retorna o centro geométrico de uma geometria. Descrição Para geometria: Retorna o comprimento cartesiano 2D se for uma LineString, MultiLineString, ST_Curve, ST_MultiCurve. Retorna 0 para geometrias areais. Use . Para tipos de geometrias, unidades para medição de comprimento estão especificadas pelo sistema de referência espacial da geometria. For geography types: computation is performed using the inverse geodesic calculation. Units of length are in meters. If PostGIS is compiled with PROJ version 4.8.0 or later, the spheroid is specified by the SRID, otherwise it is exclusive to WGS84. If use_spheroid = false, then the calculation is based on a sphere instead of a spheroid. No momento, para geometria, isto é heterômio para ST_Length2D, mas isto pode mudar para dimensões maiores. Alterações: 2.0.0 Quebrando a mudança -- nas versões anteriores aplicar isto a um MULTI/POLÍGONO de tipo de geografia lhe daria o perímetro do POLÍGONO/MULTI-POLÍGONO. Na 2.0.0 isso é alterado para retornar 0 a estar na linha com o comportamento da geometria. Por favor, utilize a ST_Perimeter se quiser o perímetro de um polígono Para a medição de geografia o padrão é a medição do esferoide. Para

usar a esfera mais rápida e menos precisa, use ST_Length(gg,false); This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.5.1 This method implements the SQL/MM specification. SQL-MM 3: 7.1.2, 9.3.4 Disponibilidade: 1.5.0 suporte para geografia foi introduzido em 1.5.

- ST_LineFromText - Faz uma geometria de uma representação WKT com a SRID dada. Se a SRID não for dada, isso leva a 0. Descrição Makes a Geometry from WKT with the given SRID. If SRID is not given, it defaults to 0. If WKT passed in is not a LINESTRING, then null is returned. OGC SPEC 3.2.6.2 - opção SRID é de uma suíte de conformação. Se você sabe que todas as suas geometrias são LINESTRINGS, é mais eficiente usar somente ST_GeomFromText. Isso só convida a ST_GeomFromText e adiciona validação extra que ela retorna uma linestring. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2 This method implements the SQL/MM specification. SQL-MM 3: 7.2.8

- ST_LineFromWKB - Faz uma LINESTRING de uma WKB com o SRID dado Descrição A função ST_LineFromWKB, pega uma representação binária bem conhecida de geometria e um sistema de referência espacial ID (SRID) e cria um exemplo do tipo apropriado de geometria - nesse caso, uma geometria LINESTRING . Essa função cumpre o papel da Fábrica de Geometria SQL. Se um SRID não estiver especificado, isso leva a 0. Retorna NULA se a entrada bytea não representa uma LINESTRING. OGC SPEC 3.2.6.2 - opção SRID é de uma suíte de conformação. Se você sabe que todas suas geometrias são LINESTRINGs, é mais eficaz usar . Essa função convida e adiciona validação extra que ela retorna uma linestring. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2 This method implements the SQL/MM specification. SQL-MM 3: 7.2.9

- ST_LinestringFromWKB - Faz uma geometria de uma WKB com o SRID dado. Descrição A função ST_LinestringFromWKB, pega uma representação binária bem conhecida de geometria e um sistema de referência espacial ID (SRID) e cria um exemplo do tipo apropriado de geometria - nesse caso, uma geometria LINESTRING . Essa função cumpre o papel da Fábrica de Geometria SQL. Se um SRID não estiver especificado, isso leva a 0. Retorna NULA se a entrada bytea não representa uma geometria LINESTRING. Isso é um heterônimo para . OGC SPEC 3.2.6.2 - o SRID opcional é da suíte de conformação. Se você sabe que todas suas geometrias são LINESTRINGs, é mais eficaz usar . Essa função convida e adiciona validação extra que ela retorna uma LINESTRING. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2 This method implements the SQL/MM specification. SQL-MM 3: 7.2.9

- ST_LocateAlong - Returns the point(s) on a geometry that match a measure value. Descrição Returns the location(s) along a measured geometry that have the given measure values. The result is a Point or MultiPoint. Polygonal inputs are not supported. If offset is provided, the result is offset to the left or right of the input line by the specified distance. A positive offset will be to the left, and a negative one to the right. Use this function only for linear geometries with an M component The semantic is specified by the ISO/IEC 13249-3 SQL/MM Spatial standard. Disponibilidade: 1.1.0 pelo nome antigo ST_Locate_Along_Measure. Alterações: 2.0.0 nas versões anteriores era chamado de ST_Locate_Along_Measure. O nome antigo foi menosprezado e será removido no futuro, mas ainda está disponível. This function supports M coordinates. This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1.13

- ST_LocateBetween - Returns the portions of a geometry that match a measure range. Descrição Retorna um valor de coleção de geometria derivado com elementos que combinam com a medida específica. Elementos polígonos não são suportados. Se um deslocamento é fornecido, o resultado será o deslocamento para a direita ou para a esquerda da linha de entrada pelo número específico de unidades. Um deslocamento positivo será para a esquerda e um negativo para a direita. Clipping a non-convex POLYGON may produce invalid geometry. The semantic is specified by the ISO/IEC 13249-3 SQL/MM Spatial standard. Disponibilidade: 1.1.0 pelo nome antigo ST_Locate_Between_Measures. Alterações: 2.0.0 nas versões anteriores era chamado de ST_Locate_Along_Measure. O nome antigo foi menosprezado e será removido no futuro, mas ainda está disponível. Enhanced: 3.0.0 - added support for POLYGON, TIN, TRIANGLE. This function supports M coordinates. This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 5.1

- ST_M - Returns the M coordinate of a Point. Descrição Retorna a coordenada M do ponto, ou NULA se não estiver disponível. Entrada deve ser um ponto. Isso não faz parte (ainda) do OGC spec, mas está listado aqui para completar a função lista do ponto coordenado extrator. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. This method implements the SQL/MM specification. This function supports 3d and will not drop the z-index.

- ST_MLineFromText - Retorna um valor específico ST_MultiLineString de uma representação WKT. Descrição Makes a Geometry from Well-Known-Text (WKT) with the given SRID. If SRID is not given, it defaults to 0. OGC SPEC 3.2.6.2 - opção SRID é de uma suíte de conformação Retorna nulo se o WKT não é uma MULTILIINESTRING Se você tem total certeza de que todas suas geometrias WKT são pontos, não use essa função. Ela é mais devagar que a ST_GeomFromText, já que adiciona um passo de validação adicional. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2 This method implements the SQL/MM specification. SQL-MM 3: 9.4.4

- ST_MPointFromText - Makes a Geometry from WKT with the given SRID. If SRID is not given, it defaults to 0. Descrição Makes a Geometry from WKT with the given SRID. If SRID is not given, it defaults to 0. OGC SPEC 3.2.6.2 - opção SRID é de uma suíte de conformação Retorna nulo se o WKT não é um MULTIPONTO Se você tem total certeza de que todas suas geometrias WKT são pontos, não use essa função. Ela é mais devagar que a ST_GeomFromText, já que adiciona um passo de validação adicional. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. 3.2.6.2 This method implements the SQL/MM specification. SQL-MM 3: 9.2.4

- ST_MPolyFromText - Makes a MultiPolygon Geometry from WKT with the given SRID. If SRID is not given, it defaults to 0. Descrição Makes a MultiPolygon from WKT with the given SRID. If SRID is not given, it defaults to 0. OGC SPEC 3.2.6.2 - opção SRID é de uma suíte de conformação Descarta um erro se o WKT não for um MULTIPOLÍGONO Se você tem total certeza de que todas suas geometrias WKT são multipolígonos, não use essa função. Ela é mais devagar que a ST_GeomFromText, já que adiciona um passo de validação adicional. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2 This method implements the SQL/MM specification. SQL-MM 3: 9.6.4

- ST_ModEdgeHeal - Heals two edges by deleting the node connecting them, modifying the first edge and deleting the second edge. Returns the id of the deleted node. Descrição Heals two edges by deleting the node connecting them, modifying the first edge and deleting the second edge. Returns the id of the deleted node. Updates all existing joined edges and relationships accordingly. Disponibilidade: 2.0 This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.9

- ST_ModEdgeSplit - Divide um limite criando um novo nó junto de um limite existente, modificando o limite original e adicionando um novo limite. Descrição Divide um limite criando um novo nó junto de um limite existente, modificando o limite original e adicionando um novo limite. Atualiza todos os limites e relacionamentos em conformidade. Retorna o identificador do novo nó adicionado. Availability: 1.1 Alterações: 2.0 - Nas versões anteriores, isto recebia o nome errado ST_ModEdgesSplit This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.9

- ST_MoveIsoNode - Moves an isolated node in a topology from one point to another. If new apoint geometry exists as a node an error is thrown. Returns description of move. Descrição Move um nó isolado em uma topologia de um ponto para outro. Se nova geometria apoint existe como um nó, um erro é lançado. If any arguments are null, the apoint is not a point, the existing node is not isolated (is a start or end point of an existing edge), new node location intersects an existing edge (even at the end points) or the new location is in a different face (since 3.2.0) then an exception is thrown. Se o sistema de referência espacial (srid) da geometria pontual não for o mesmo da topologia, uma exceção é lançada. Disponibilidade: 2.0.0 Enhanced: 3.2.0 ensures the nod cannot be moved in a different face This method implements the SQL/MM specification. SQL-MM: Topo-Net Routines: X.3.2

- ST_NewEdgeHeal - Heals two edges by deleting the node connecting them, deleting both edges, and replacing them with an edge whose direction is the same as the first edge provided. Descrição Heals two edges by deleting the node connecting them, deleting both edges, and replacing them with an edge whose direction is the same as the first edge provided. Returns the id of the new edge replacing the healed ones. Updates all existing joined edges and relationships accordingly. Disponibilidade: 2.0 This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.9

- ST_NewEdgesSplit - Divide um limite criando um novo nó ao longo do limite existente, deletando o limite original e substituindo-o por dois novos. Retorna a id do novo nó criado que integra os novos limites. Descrição Divide um limite com uma id limite anedge criando um novo nó com uma localização de ponto apoint junto co i limite atual, deletando o limite original e substituindo-o por dois novos. Retorna a id do novo nó criado que se une aos novos limites. Atualiza todos os limites unidos e relacionamentos em conformidade. Se o sistema de referência espacial (srid) da geometria pontual não é o mesmo que a topologia, o apoint não é uma geometria pontual, o ponto é nulo, o ponto já existe como um nó, o limite não corresponde a um limite existente ou o ponto não está dentro do limite, então, uma exceção é aberta. Availability: 1.1 This method implements the SQL/MM specification. SQL-MM: Topo-Net Routines: X.3.8

- ST_NumGeometries - Retorna o número de pontos em uma geometria. Funciona para todas as geometrias. Descrição Retorna o número de geometrias. Se a geometria é uma GEOMETRYCOLLECTION (ou MULTI*), retorna o número de geometria, para geometrias únicas retornará 1, senão retorna NULO. Melhorias: 2.0.0 suporte para superfícies poliédricas, triângulos e TIN introduzido. Alterações: 2.0.0 Em versões anteriores retornaria NULO se a geometria não fosse do tipo coleção/MULTI. 2.0.0+ agora retorna 1 para geometrias únicas ex: POLÍGONO, LINESTRING, PONTO. This method implements the SQL/MM specification. SQL-MM 3: 9.1.4 This function supports 3d and will not drop the z-index. This function supports Polyhedral surfaces. This function supports Triangles and Triangulated Irregular Network Surfaces (TIN).

- ST_NumInteriorRings - Retorna o número de anéis interiores de um polígono. Descrição Retorna o número de anéis interiores de um polígono. Retorna NULO se a geometria não for um polígono. This method implements the SQL/MM specification. SQL-MM 3: 8.2.5 Alterações: 2.0.0 - nas versões anteriores isso permitiria um MULTIPOLÍGONO, retornando o número de anéis interiores do primeiro POLÍGONO.

- ST_NumPatches - Retorna o número de faces em uma superfícies poliédrica. Retornará nulo para geometrias não poliédricas. Descrição Retorna o número de faces em uma superfície poliédrica. Retornará nulo para geometrias não poliédricas. Isso é um heterônimo para ST_NumGeometries para suportar a nomeação MM. É mais rápido utilizar ST_NumGeometries se você não se importa com a convenção MM. Disponibilidade: 2.0.0 This function supports 3d and will not drop the z-index. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. This method implements the SQL/MM specification. SQL-MM ISO/IEC 13249-3: 8.5 This function supports Polyhedral surfaces.

- ST_NumPoints - Retorna o número de pontos em um valor ST_LineString ou ST_CircularString. Descrição Retorna o número de pontos em um valor ST_LineString ou ST_CircularString. Anteriores a 1.4 só funcionam com Linestrings como as specs declaram. A partir de 1.4 isso é um heterônimo para ST_NPoints, que retorna o número de vértices apenas para as line strings. Considere utilizar ST_NPoints que tem vários objetivos e funciona com vários tipos de geometrias. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. This method implements the SQL/MM specification. SQL-MM 3: 7.2.4

- ST_OrderingEquals - Tests if two geometries represent the same geometry and have points in the same directional order Description ST_OrderingEquals compares two geometries and returns t (TRUE) if the geometries are equal and the coordinates are in the same order; otherwise it returns f (FALSE). This function is implemented as per the ArcSDE SQL specification rather than SQL-MM. http://edndoc.esri.com/arcsde/9.1/sql_api/sqlapi3.htm#ST_OrderingEquals This method implements the SQL/MM specification. SQL-MM 3: 5.1.43

- ST_Overlaps - Tests if two geometries have the same dimension and intersect, but each has at least one point not in the other Description Returns TRUE if geometry A and B "spatially overlap". Two geometries overlap if they have the same dimension, their interiors intersect in that dimension. and each has at least one point inside the other (or equivalently, neither one covers the other). The overlaps relation is symmetric and irreflexive. In mathematical terms: ST_Overlaps(A, B) $\Leftrightarrow$ (dim(A) = dim(B) = dim(Int(A) $\bigcap$ Int(B))) $\wedge$ (A $\bigcap$ B $\neq$ A) $\wedge$ (A $\bigcap$ B $\neq$ B) This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries. To avoid index use, use the function _ST_Overlaps. Performed by the GEOS module Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION NOTE: this is the "allowable" version that returns a boolean, not an integer. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.2 // s2.1.13.3 This method implements the SQL/MM specification. SQL-MM 3: 5.1.32

- ST_PatchN - Retorna o tipo de geometria de valor ST_Geometry. Descrição > Retorna a geometria (face) de 1-base Nth se a geometria é POLYHEDRALSURFACE, POLYHEDRALSURFACEM. Senão, retorna NULA. Retorna a mesma resposta como ST_GeometryN para superfícies poliédricas. Utilizar ST_GemoetryN é mais rápido. Index é 1-base. Se você quiser extrair todas as geometrias, de uma geometria, ST_Dump é mais eficiente. Disponibilidade: 2.0.0 This method implements the SQL/MM specification. SQL-MM ISO/IEC 13249-3: 8.5 This function supports 3d and will not drop the z-index. This function supports Polyhedral surfaces.

- ST_Perimeter - Returns the length of the boundary of a polygonal geometry or geography. Descrição Retorna o perímetro 2D da geometria/geografia se for uma ST_Surface, ST_MultiSurface (Polygon, MultiPolygon). Retorna 0 para geometrias não areais. Para geometrias lineares, use . Para tipos de geometria, unidades para medição de perímetro estão especificadas pelo sistema de referência espacial da geometria. For geography types, the calculations are performed using the inverse geodesic problem, where perimeter units are in meters. If PostGIS is compiled with PROJ version 4.8.0 or later, the spheroid is specified by the SRID, otherwise it is exclusive to WGS84. If use_spheroid = false, then calculations will approximate a sphere instead of a spheroid. No momento isto é um heterônimo para ST_Perimeter2D, mas pode ser alterado para suportar dimensões maiores. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.5.1 This method implements the SQL/MM specification. SQL-MM 3: 8.1.3, 9.5.4 Disponibilidade 2.0.0: Suporte para geografia foi introduzido

- ST_Point - Creates a Point with X, Y and SRID values. Descrição Returns a Point with the given X and Y coordinate values. This is the SQL-MM equivalent for that takes just X and Y. For geodetic coordinates, X is longitude and Y is latitude Enhanced: 3.2.0 srid as an extra optional argument was added. Older installs require combining with ST_SetSRID to mark the srid on the geometry. This method implements the SQL/MM specification. SQL-MM 3: 6.1.2

- ST_PointFromText - Faz um ponto de um WKT com o SRID dado. Se o SRID não for dado, isso leva a desconhecido. Descrição Constructs a PostGIS ST_Geometry point object from the OGC Well-Known text representation. If SRID is not

given, it defaults to unknown (currently 0). If geometry is not a WKT point representation, returns null. If completely invalid WKT, then throws an error. Existem 2 variantes da função ST_PointFromText, a primeira não pega nenhuma SRID e retorna uma geometria sem sistema de referência espacial definido. A segunda, pega uma id referência espacial como o segundo argumento e retorna uma ST_Geometry que inclui esse srid como parte dos seus metadados. O srid deve ser definido na spatial_ref_sys table. Se você tem total certeza de que todas suas geometrias WKT são pontos, não use essa função. Ela é mais devagar que a ST_GeomFromText, já que adiciona um passo de validação adicional. Se você está construindo pontos de coordenadas long lat e se importa mais com apresentação e precisão do que com concordância OGC, use: ou . This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2 - opção SRID é da suíte de conformidade. This method implements the SQL/MM specification. SQL-MM 3: 6.1.8

- ST_PointFromWKB - Faz uma geometria a partir de um WKB com o SRID dado Descrição A função ST_PointFromWKB, pega uma representação binária bem conhecida de geometria e um sistema de referência espacial ID (SRID) e cria um exemplo do tipo apropriado de geometria - nesse caso, uma geometria PONTO . Essa função cumpre o papel da Fábrica de Geometria SQL. Se uma SRID não for especificada, leva a 0. NULO é retornado se a entrada bytea não representar uma PONTO geometria. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s3.2.7.2 This method implements the SQL/MM specification. SQL-MM 3: 6.1.9 This function supports 3d and will not drop the z-index. This method supports Circular Strings and Curves.

- ST_PointN - Retorna o número de pontos em um valor ST_LineString ou ST_CircularString. Descrição Retorna o ponto Nth na primeira linestring ou linestring circular na geometria. Valores negativos são contados tardiamente do fim da linestring, tornando o ponto -1 o último ponto. Retorna NULA se não há uma linestring na geometria. O Index é 1-base como para OGC specs desde a versão 0.8.0. Indexing atrasado (negativo) não está nas versões OGC anteriores implementadas com 0-base. Se você quiser o ponto nth de cada line string em uma multilinestring, utilize em conjunção com ST_Dump This method implements the OGC Simple Features Implementation Specification for SQL 1.1. This method implements the SQL/MM specification. SQL-MM 3: 7.2.5, 7.3.5 This function supports 3d and will not drop the z-index. This method supports Circular Strings and Curves. Alterações: 2.0.0 não funciona mais com geometrias multilinestrings únicas. Em verões mais antigas do PostGIS -- uma única linha multilinestring trabalharia normalmente e retornaria o ponto inicial. Na 2.0.0 só retorna NULA como qualquer outra multilinestring. Alterações: 2.3.0 : indexing negativo disponível (-1 é o último ponto)

- ST_PointOnSurface - Computes a point guaranteed to lie in a polygon, or on a geometry. Descrição Returns a POINT which is guaranteed to lie in the interior of a surface (POLYGON, MULTIPOLYGON, and CURVED POLYGON). In PostGIS this function also works on line and point geometries. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s3.2.14.2 // s3.2.18.2 This method implements the SQL/MM specification. SQL-MM 3: 8.1.5, 9.5.6. The specifications define ST_PointOnSurface for surface geometries only. PostGIS extends the function to support all common geometry types. Other databases (Oracle, DB2, ArcSDE) seem to support this function only for surfaces. SQL Server 2008 supports all common geometry types. This function supports 3d and will not drop the z-index.

- ST_Polygon - Creates a Polygon from a LineString with a specified SRID. Descrição Returns a polygon built from the given LineString and sets the spatial reference system from the srid. ST_Polygon is similar to Variant 1 with the addition of setting the SRID. , , Essa função não aceitará uma MULTILINESTRING. Use ou para gerar line strings. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. This method implements the SQL/MM specification. SQL-MM 3: 8.3.2 This function supports 3d and will not drop the z-index.

- ST_PolygonFromText - Makes a Geometry from WKT with the given SRID. If SRID is not given, it defaults to 0. Descrição Makes a Geometry from WKT with the given SRID. If SRID is not given, it defaults to 0. Returns null if WKT is not a polygon. OGC SPEC 3.2.6.2 - opção SRID é de uma suíte de conformação Se você tem total certeza de que todas suas geometrias WKT são polígonos, não use essa função. Ela é mais devagar que a ST_GeomFromText, já que adiciona um passo de validação adicional. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s3.2.6.2 This method implements the SQL/MM specification. SQL-MM 3: 8.3.6

- ST_Relate - Tests if two geometries have a topological relationship matching an Intersection Matrix pattern, or computes their Intersection Matrix Description These functions allow testing and evaluating the spatial (topological) relationship between two geometries, as defined by the Dimensionally Extended 9-Intersection Model (DE-9IM). The DE-9IM is specified as a 9-element matrix indicating the dimension of the intersections between the Interior, Boundary and Exterior of two geometries. It is represented by a 9-character text string using the symbols 'F', '0', '1', '2' (e.g. 'FF1FF0102'). A specific kind of spatial relationship can be tested by matching the intersection matrix to an intersection matrix pattern. Patterns can include the additional symbols 'T' (meaning "intersection is non-empty") and '*' (meaning "any value"). Common spatial relationships are provided by the named functions , , , , , , , , , and . Using an explicit pattern allows testing multiple conditions of intersects, crosses, etc in one step. It also allows testing spatial relationships which do not have a named spatial relationship

function. For example, the relationship "Interior-Intersects" has the DE-9IM pattern T********, which is not evaluated by any named predicate. For more information refer to . Variant 1: Tests if two geometries are spatially related according to the given intersectionMatrixPattern. Unlike most of the named spatial relationship predicates, this does NOT automatically include an index call. The reason is that some relationships are true for geometries which do NOT intersect (e.g. Disjoint). If you are using a relationship pattern that requires intersection, then include the && index call. It is better to use a named relationship function if available, since they automatically use a spatial index where one exists. Also, they may implement performance optimizations which are not available with full relate evalation. Variant 2: Returns the DE-9IM matrix string for the spatial relationship between the two input geometries. The matrix string can be tested for matching a DE-9IM pattern using . Variant 3: Like variant 2, but allows specifying a Boundary Node Rule. A boundary node rule allows finer control over whether the endpoints of MultiLineStrings are considered to lie in the DE-9IM Interior or Boundary. The boundaryNodeRule values are: 1: OGC-Mod2 - line endpoints are in the Boundary if they occur an odd number of times. This is the rule defined by the OGC SFS standard, and is the default for ST_Relate. 2: Endpoint - all endpoints are in the Boundary. 3: MultivalentEndpoint - endpoints are in the Boundary if they occur more than once. In other words, the boundary is all the "attached" or "inner" endpoints (but not the "unattached/outer" ones). 4: MonovalentEndpoint - endpoints are in the Boundary if they occur only once. In other words, the boundary is all the "unattached" or "outer" endpoints. This function is not in the OGC spec, but is implied. see s2.1.13.2 This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.2 // s2.1.13.3 This method implements the SQL/MM specification. SQL-MM 3: 5.1.25 Performed by the GEOS module Enhanced: 2.0.0 - added support for specifying boundary node rule. Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION

- ST_RemEdgeModFace - Removes an edge, and if the edge separates two faces deletes one face and modifies the other face to cover the space of both. Descrição Removes an edge, and if the removed edge separates two faces deletes one face and modifies the other face to cover the space of both. Preferentially keeps the face on the right, to be consistent with . Returns the id of the face which is preserved. Atualiza todos os limites existentes e relacionamentos em conformidade. Refuses to remove an edge participating in the definition of an existing TopoGeometry. Refuses to heal two faces if any TopoGeometry is defined by only one of them (and not the other). Se qualquer argumento for nulo, o limite dado é desconhecido (deve existir na table edge do esquema de topologia), o nome da topologia é inválido, logo, um erro é lançado. Disponibilidade: 2.0 This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.15

- ST_RemEdgeNewFace - Remove um limite e, se o limite removido separava duas faces, deleta as faces originais e as substitui por uma nova face. Descrição Remove um limite e, se o limite removido separava duas faces, deleta as faces originais e as substitui por uma nova face. Retorna a id de uma face nova criada ou NULA, se nenhuma face nova for criada. Nenhuma face nova é criada quando o limite removido está pendurado, isolado ou confinado na face universal (possivelmente fazendo a inundação universal dentro da face no outro lado). Atualiza todos os limites existentes e relacionamentos em conformidade. Refuses to remove an edge participating in the definition of an existing TopoGeometry. Refuses to heal two faces if any TopoGeometry is defined by only one of them (and not the other). Se qualquer argumento for nulo, o limite dado é desconhecido (deve existir na table edge do esquema de topologia), o nome da topologia é inválido, logo, um erro é lançado. Disponibilidade: 2.0 This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X.3.14

- ST_RemoveIsoEdge - Removes an isolated edge and returns description of action. If the edge is not isolated, then an exception is thrown. Descrição Removes an isolated edge and returns description of action. If the edge is not isolated, then an exception is thrown. Availability: 1.1 This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X+1.3.3

- ST_RemoveIsoNode - Remove um nó isolado e retorna descrição de ação. Se o nó não for isolado (for começo ou fim de um limite), então, uma exceção é lançada. Descrição Remove um nó isolado e retorna descrição de ação. Se o nó não for isolado (for começo ou fim de um limite), então, uma exceção é lançada. Availability: 1.1 This method implements the SQL/MM specification. SQL-MM: Topo-Geo and Topo-Net 3: Routine Details: X+1.3.3

- ST_SRID - Returns the spatial reference identifier for a geometry. Description Returns the spatial reference identifier for the ST_Geometry as defined in spatial_ref_sys table. spatial_ref_sys table is a table that catalogs all spatial reference systems known to PostGIS and is used for transformations from one spatial reference system to another. So verifying you have the right spatial reference system identifier is important if you plan to ever transform your geometries. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.1 This method implements the SQL/MM specification. SQL-MM 3: 5.1.5 This method supports Circular Strings and Curves.

- ST_StartPoint - Returns the first point of a LineString. Descrição Retorna ao último ponto de uma LINESTRING ou CIRCULARLINESTRING geometria como um PONTO ou NULO se o parâmetro de entrada não é uma LINESTRING. This method implements the SQL/MM specification. SQL-MM 3: 7.1.3 This function supports 3d and will not drop the z-index. This method supports Circular Strings and Curves. Enhanced: 3.2.0 returns a point for all geometries. Prior behavior returns

NULLs if input was not a LineString. Alterações: 2.0.0 não funciona mais com geometrias de multilinestrings. Em verões mais antigas do PostGIS -- uma linha multilinestring sozinha trabalharia normalmente com essa função e voltaria o ponto de início. Na 2.0.0 ela retorna NULA como qualquer outra multilinestring. O antigo comportamento não foi uma característica documentada, mas as pessoas que consideravam que tinham seus dados armazenados como uma LINESTRING, agora podem experimentar essas que retornam NULAS em 2.0.

- ST_SymDifference - Computes a geometry representing the portions of geometries A and B that do not intersect. Description Returns a geometry representing the portions of geonetries A and B that do not intersect. This is equivalent to ST_Union(A,B) - ST_Intersection(A,B). It is called a symmetric difference because ST_SymDifference(A,B) = ST_SymDifference(B,A). If the optional gridSize argument is provided, the inputs are snapped to a grid of the given size, and the result vertices are computed on that same grid. (Requires GEOS-3.9.0 or higher) Performed by the GEOS module Enhanced: 3.1.0 accept a gridSize parameter. Requires GEOS >= 3.9.0 to use the gridSize parameter This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.3 This method implements the SQL/MM specification. SQL-MM 3: 5.1.21 This function supports 3d and will not drop the z-index. However, the result is computed using XY only. The result Z values are copied, averaged or interpolated.

- ST_Touches - Tests if two geometries have at least one point in common, but their interiors do not intersect Description Returns TRUE if A and B intersect, but their interiors do not intersect. Equivalently, A and B have at least one point in common, and the common points lie in at least one boundary. For Point/Point inputs the relationship is always FALSE, since points do not have a boundary. In mathematical terms: ST_Touches(A, B) $\Leftrightarrow$ (Int(A) $\bigcap$ Int(B) $\neq \varnothing$) $\wedge$ (A $\bigcap$ B $\neq \varnothing$) This relationship holds if the DE-9IM Intersection Matrix for the two geometries matches one of: FT******* F**T***** F***T**** This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries. To avoid using an index, use _ST_Touches instead. Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.2 // s2.1.13.3 This method implements the SQL/MM specification. SQL-MM 3: 5.1.28

- ST_Transform - Return a new geometry with coordinates transformed to a different spatial reference system. Description Returns a new geometry with its coordinates transformed to a different spatial reference system. The destination spatial reference to_srid may be identified by a valid SRID integer parameter (i.e. it must exist in the spatial_ref_sys table). Alternatively, a spatial reference defined as a PROJ.4 string can be used for to_proj and/or from_proj, however these methods are not optimized. If the destination spatial reference system is expressed with a PROJ.4 string instead of an SRID, the SRID of the output geometry will be set to zero. With the exception of functions with from_proj, input geometries must have a defined SRID. ST_Transform is often confused with . ST_Transform actually changes the coordinates of a geometry from one spatial reference system to another, while ST_SetSRID() simply changes the SRID identifier of the geometry. ST_Transform automatically selects a suitable conversion pipeline given the source and target spatial reference systems. To use a specific conversion method, use . Requires PostGIS be compiled with PROJ support. Use to confirm you have PROJ support compiled in. If using more than one transformation, it is useful to have a functional index on the commonly used transformations to take advantage of index usage. Prior to 1.3.4, this function crashes if used with geometries that contain CURVES. This is fixed in 1.3.4+ Enhanced: 2.0.0 support for Polyhedral surfaces was introduced. Enhanced: 2.3.0 support for direct PROJ.4 text was introduced. This method implements the SQL/MM specification. SQL-MM 3: 5.1.6 This method supports Circular Strings and Curves. This function supports Polyhedral surfaces.

- ST_Union - Computes a geometry representing the point-set union of the input geometries. Description Unions the input geometries, merging geometry to produce a result geometry with no overlaps. The output may be an atomic geometry, a MultiGeometry, or a Geometry Collection. Comes in several variants: Two-input variant: returns a geometry that is the union of two input geometries. If either input is NULL, then NULL is returned. Array variant: returns a geometry that is the union of an array of geometries. Aggregate variant: returns a geometry that is the union of a rowset of geometries. The ST_Union() function is an "aggregate" function in the terminology of PostgreSQL. That means that it operates on rows of data, in the same way the SUM() and AVG() functions do and like most aggregates, it also ignores NULL geometries. See for a non-aggregate, single-input variant. The ST_Union array and set variants use the fast Cascaded Union algorithm described in http://blog.cleverelephant.ca/2009/01/must-faster-unions-in-postgis-14.html A gridSize can be specified to work in fixed-precision space. The inputs are snapped to a grid of the given size, and the result vertices are computed on that same grid. (Requires GEOS-3.9.0 or higher) may sometimes be used in place of ST_Union, if the result is not required to be non-overlapping. ST_Collect is usually faster than ST_Union because it performs no processing on the collected geometries. Performed by the GEOS module. ST_Union creates MultiLineString and does not sew LineStrings into a single LineString. Use to sew LineStrings. NOTE: this function was formerly called GeomUnion(), which was renamed from "Union" because UNION is an SQL reserved word. Enhanced: 3.1.0 accept a gridSize parameter. Requires GEOS >= 3.9.0 to use the gridSize parameter Changed: 3.0.0 does not depend on SFCGAL. Availability: 1.4.0 - ST_Union was enhanced. ST_Union(geomarray)

was introduced and also faster aggregate collection in PostgreSQL. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.3 Aggregate version is not explicitly defined in OGC SPEC. This method implements the SQL/MM specification. SQL-MM 3: 5.1.19 the z-index (elevation) when polygons are involved. This function supports 3d and will not drop the z-index. However, the result is computed using XY only. The result Z values are copied, averaged or interpolated.

- ST_Volume - Computa o volume de um sólido 3D. Se aplicado a geometrias com superfícies (mesmo fechadas), irão retornar 0. Descrição Disponibilidade: 2.2.0 This method needs SFCGAL backend. This function supports 3d and will not drop the z-index. This function supports Polyhedral surfaces. This function supports Triangles and Triangulated Irregular Network Surfaces (TIN). This method implements the SQL/MM specification. SQL-MM IEC 13249-3: 9.1 (same as ST_3DVolume)

- ST_WKBToSQL - Retorna um valor ST_Geometry especifico da representação de texto binário bem conhecida (WKB). Isso é um heterônimo para ST_GeomFromWKB que não pega nenhum srid Descrição This method implements the SQL/MM specification. SQL-MM 3: 5.1.36

- ST_WKTToSQL - Retorna um valor ST_Geometry especifico da representação de texto estendida bem conhecida (EWKT). Isso é um heterônimo para ST_GeomFromText Descrição This method implements the SQL/MM specification. SQL-MM 3: 5.1.34

- ST_Within - Tests if every point of A lies in B, and their interiors have a point in common Description Returns TRUE if geometry A is within geometry B. A is within B if and only if all points of A lie inside (i.e. in the interior or boundary of) B (or equivalently, no points of A lie in the exterior of B), and the interiors of A and B have at least one point in common. For this function to make sense, the source geometries must both be of the same coordinate projection, having the same SRID. In mathematical terms: ST_Within(A, B) $\Leftrightarrow$ (A $\bigcap$ B = A) $\wedge$ (Int(A) $\bigcap$ Int(B) $\neq \varnothing$) The within relation is reflexive: every geometry is within itself. The relation is antisymmetric: if ST_Within(A,B) = true and ST_Within(B,A) = true, then the two geometries must be topologically equal (ST_Equals(A,B) = true). ST_Within is the converse of . So, ST_Within(A,B) = ST_Contains(B,A). Because the interiors must have a common point, a subtlety of the definition is that lines and points lying fully in the boundary of polygons or lines are not within the geometry. For further details see Subtleties of OGC Covers, Contains, Within. The predicate provides a more inclusive relationship. This function automatically includes a bounding box comparison that makes use of any spatial indexes that are available on the geometries. To avoid index use, use the function _ST_Within. Performed by the GEOS module Enhanced: 2.3.0 Enhancement to PIP short-circuit for geometry extended to support MultiPoints with few points. Prior versions only supported point in polygon. Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION Do not use this function with invalid geometries. You will get unexpected results. NOTE: this is the "allowable" version that returns a boolean, not an integer. This method implements the OGC Simple Features Implementation Specification for SQL 1.1. s2.1.1.2 // s2.1.13.3 - a.Relate(b, 'T*F**F***') This method implements the SQL/MM specification. SQL-MM 3: 5.1.30

- ST_X - Returns the X coordinate of a Point. Descrição Retorna a coordenada X do ponto, ou NULA se não estiver disponível. Entrada deve ser um ponto. To get the minimum and maximum X value of geometry coordinates use the functions and . This method implements the SQL/MM specification. SQL-MM 3: 6.1.3 This function supports 3d and will not drop the z-index.

- ST_Y - Returns the Y coordinate of a Point. Descrição Retorna a coordenada Y do ponto, ou NULA se não estiver disponível. Entrada deve ser um ponto. To get the minimum and maximum Y value of geometry coordinates use the functions and . This method implements the OGC Simple Features Implementation Specification for SQL 1.1. This method implements the SQL/MM specification. SQL-MM 3: 6.1.4 This function supports 3d and will not drop the z-index.

- ST_Z - Returns the Z coordinate of a Point. Descrição Retorna a coordenada Z do ponto, ou NULA se não estiver disponível. Entrada deve ser um ponto. To get the minimum and maximum Z value of geometry coordinates use the functions and . This method implements the SQL/MM specification. This function supports 3d and will not drop the z-index.

- TG_ST_SRID - Returns the spatial reference identifier for a topogeometry. Descrição Returns the spatial reference identifier for the ST_Geometry as defined in spatial_ref_sys table. spatial_ref_sys table is a table that catalogs all spatial reference systems known to PostGIS and is used for transformations from one spatial reference system to another. So verifying you have the right spatial reference system identifier is important if you plan to ever transform your geometries. Availability: 3.2.0 This method implements the SQL/MM specification. SQL-MM 3: 14.1.5

12.4 PostGIS Geography Support Functions

The functions and operators given below are PostGIS functions/operators that take as input or return as output a geography data type object.

Note
Functions with a (T) are not native geodetic functions, and use a ST_Transform call to and from geometry to do the operation. As a result, they may not behave as expected when going over dateline, poles, and for large geometries or geometry pairs that cover more than one UTM zone. Basic transform - (favoring UTM, Lambert Azimuthal (North/South), and falling back on mercator in worst case scenario)

- ST_Area - Retorna o centro geométrico de uma geometria.
- ST_AsBinary - Return the OGC/ISO Well-Known Binary (WKB) representation of the geometry/geography without SRID meta data.
- ST_AsEWKT - Retorna a representação de texto bem conhecida (WKT) da geometria com os meta dados SRID.
- ST_AsGML - Retorna a geometria como uma versão GML com 2 ou 3 elementos.
- ST_AsGeoJSON - Return a geometry as a GeoJSON element.
- ST_AsKML - Retorna a geometria como uma versão GML com 2 ou 3 elementos.
- ST_AsSVG - Returns SVG path data for a geometry.
- ST_AsText - Retorna a representação de texto bem conhecida (WKT) da geometria/geografia sem os meta dados do SRID.
- ST_Azimuth - Retorna a menor linha 2-dimensional entre duas geometrias
- ST_Buffer - Computes a geometry covering all points within a given distance from a geometry.
- ST_Centroid - Retorna o centro geométrico de uma geometria.
- ST_ClosestPoint - Returns the 2D point on g1 that is closest to g2. This is the first point of the shortest line from one geometry to the other.
- ST_CoveredBy - Tests if every point of A lies in B
- ST_Covers - Tests if every point of B lies in A
- ST_DWithin - Tests if two geometries are within a given distance
- ST_Distance - Retorna a linha 3-dimensional mais longa entre duas geometrias
- ST_GeogFromText - Retorna um valor de geografia específico de uma representação bem conhecida de texto ou estendida (WKT).
- ST_GeogFromWKB - Cria uma ocasião geografia de uma geometria binária bem conhecida (WKB) ou binário estendido bem conhecido (EWKB).
- ST_GeographyFromText - Retorna um valor de geografia específico de uma representação bem conhecida de texto ou estendida (WKT).
- = - Returns TRUE if the coordinates and coordinate order geometry/geography A are the same as the coordinates and coordinate order of geometry/geography B.
- ST_Intersection - Computes a geometry representing the shared portion of geometries A and B.
- ST_Intersects - Tests if two geometries intersect (they have at least one point in common)
- ST_Length - Retorna o centro geométrico de uma geometria.

- ST_LineInterpolatePoint - Returns a point interpolated along a line at a fractional location.
- ST_LineInterpolatePoints - Returns points interpolated along a line at a fractional interval.
- ST_LineLocatePoint - Returns the fractional location of the closest point on a line to a point.
- ST_LineSubstring - Returns the part of a line between two fractional locations.
- ST_Perimeter - Returns the length of the boundary of a polygonal geometry or geography.
- ST_Project - Returns a point projected from a start point by a distance and bearing (azimuth).
- ST_Segmentize - Returns a modified geometry/geography having no segment longer than a given distance.
- ST_ShortestLine - Retorna a menor linha 2-dimensional entre duas geometrias
- ST_Summary - Retorna um texto resumo dos conteúdos da geometria.
- <-> - Retorna a distância 2D entre A e B.
- && - Retorna VERDADE se a caixa limitadora 2D de A intersecta a caixa limitadora 2D de B.

12.5 PostGIS Raster Support Functions

The functions and operators given below are PostGIS functions/operators that take as input or return as output a raster data type object. Listed in alphabetical order.

- Caixa3D - Retorna a representação da caixa 3d da caixa encerrada do raster.
- @ - Retorna VERDADEse a caixa limitadora de A estiver contida pela de B. Utiliza precisão dupla de caixa limitadora.
- ~ - Retorna TRUE se a caixa delimitadora de A estiver contida na do B. Utiliza caixa delimitadora de precisão dupla.
- = - Retorna VERDADEse a caixa limitadora de A for a mesma de B. Utiliza precisão dupla de caixa limitadora.
- && - Retorna VERDADE se a caixa limitadora de A intersecta a caixa limitadora de B.
- &< - Retorna VERDADE se uma caixa limitadora de A está à esquerda da de B.
- &> - Retorna VERDADE se uma caixa limitadora de A está à direita da de B.
- ~= - Retorna VERDADE se a caixa limitadora de A é a mesma de B.
- ST_Retile - Retorna um conjunto de tiles configuradas de uma cobertura raster aleatória.
- ST_AddBand - Retorna um raster com nova banda(s) do tipo dado adicionado com o valor inicial com a localização do índice. Se nenhum índice for especificado, a banda é adicionada ao final.
- ST_AsBinary/ST_AsWKB - Return the Well-Known Binary (WKB) representation of the raster.
- ST_AsGDALRaster - Return the raster tile in the designated GDAL Raster format. Raster formats are one of those supported by your compiled library. Use ST_GDALDrivers() to get a list of formats supported by your library.
- ST_AsHexWKB - Return the Well-Known Binary (WKB) in Hex representation of the raster.
- ST_AsJPEG - Retorna as bandas tile raster selecionadas como uma única Joint Photographic Exports Group (JPEG) image (byte arranjo). Se nenhuma banda for especificada e 1 ou mais que 3 bandas, então somente a primeira banda é usada. Se somente 3 bandas, então todas as 3 bandas serão usadas para mapear par RGB.
- ST_AsPNG - Retorna as bandas tile raster selecionadas como um gráfico de rede portátil (PNG) imagem (byte array). Se as bandas raster 1, 3 ou 4 e nenhum banda for especificado, então todas as bandas são usadas. Se mais 2 ou mais que 4 bandas e nenhuma banda forem especificadas, então somente a banda 1 é usada. As bandas são mapeadas para espeço RGB ou RGBA.
- ST_AsRaster - Converte uma geometria PostGIS para um raster PostGIS.

- ST_AsTIFF - Return the raster selected bands as a single TIFF image (byte array). If no band is specified or any of specified bands does not exist in the raster, then will try to use all bands.
- ST_Aspect - Retorna o aspecto (em graus) de uma banda raster de elevação. Útil para analisar terrenos.
- ST_Band - Retorna uma ou mais bandas de um raster existente como um novo raster. Útil para a construção de novos rasters a partir de rasters existentes.
- ST_BandFileSize - Returns the file size of a band stored in file system. If no bandnum specified, 1 is assumed.
- ST_BandFileTimestamp - Returns the file timestamp of a band stored in file system. If no bandnum specified, 1 is assumed.
- ST_BandIsNoData - Retorna verdadeiro se a banda estiver repleta somente de valores nodata.
- ST_BandMetaData - Retorna os metadados básicos para uma banda raster especificada. banda número 1 é assumida se nenhuma for especificada.
- ST_BandNoDataValue - Retorna o valor em uma dada banda que não representa nenhum valor. Se nenhuma banda número 1 for assumida.
- ST_BandPath - Retorna o caminho do arquivo do sistema para uma banda armazenada em um sistema de arquivos. Se nenhum número de banda for especificado, usa-se 1.
- ST_BandPixelType - Retorna o tipo pixel para uma dada banda. Se nenhum número de banda for especificado, usa-se 1.
- ST_Clip - Retorna o raster suprimido pela geometria de entrada. Se o número de banda não for especificado, todas as bandas são processadas. Se crop não for especificado ou for VERDADE, o raster de saída é cortado.
- ST_ColorMap - Cria um novo raster de até quatro bandas 8BUI (grayscale, RGB, RGBA) do raster fonte e uma banda específica. A banda 1 usada se não especificado.
- ST_Contains - Retorna verdade se nenhum ponto do raster rasteB estiver no exterior do raster rastA e pelo menos um ponto do interior do rastB estiver no interior do rastA.
- ST_ContainsProperly - Retorna verdade se o rastB intersectar o interior do rastA, mas não o limite ou exterior do ratA.
- ST_Count - Generates a set of vector contours from the provided raster band, using the GDAL contouring algorithm.
- ST_ConvexHull - Retorna o casco convexo da geometria do raster incluindo valores iguais ao BandNoDataValue. Para rasters com formas normais e não desviadas, o resultado é o mesmo que ST_Envelope, então só é útil para rasters com formas irregulares ou desviados.
- ST_Count - Retorna o número de pixeis em uma banda dada de um raster ou cobertura raster. Se nenhuma banda for especificada, o padrão é usar a banda 1. Se exclude_nodata_value for verdade, contará somente pixeis que não são iguais ao valor nodata.
- ST_CountAgg - Agregado. Retorna o número de pixeis em uma banda dada de um raster ou cobertura raster. Se nenhuma banda for especificada, o padrão é usar a banda 1. Se exclude_nodata_value for verdade, contará somente pixeis que são diferentes ao valor NODATA.
- ST_CoveredBy - Retorna verdade se nenhum ponto do rastA estiver de fora do rastB.
- ST_Covers - Retorna verdade se nenhum ponto do rastB estiver de fora do rastA.
- ST_DFullyWithin - Retorna verdade se os rasters rastA e rastB estiverem completamente dentro da distância especificada de cada um.
- ST_DWithin - Retorna verdade se os rasters rastA e rastB estiverem dentro da distância especificada de cada um.
- ST_Disjoint - Retorna verdade se raster rastA não intersectar espacialmente com o rastB.
- ST_DumpAsPolygons - Retorna um conjunto de linhas geomval (geom,val), de uma dada banda raster. Se nenhum número de banda for especificado, o número de banda torna-se 1.
- ST_DumpValues - Obtenha os valores da banda específica como um arranjo 2-dimensional.

- ST_Envelope - Retorna a representação de polígono da extensão do raster.
- ST_FromGDALRaster - Retorna um raster de um arquivo raster GDAL suportado.
- ST_GeoReference - Retorna os metadados georreferenciados no formato GDAL ou ESRI como é comumente visto em um arquivo mundo. O padrão é GDAL.
- ST_Grayscale - Creates a new one-8BUI band raster from the source raster and specified bands representing Red, Green and Blue
- ST_HasNoBand - Retorna verdade se não existirem bandas com números dados. Se nenhum número de banda for especificado, então assume-se a banda 1.
- ST_Height - Retorna a altura do raster em pixeis.
- ST_HillShade - Retorna a iluminação hipotética de uma banda raster de elevação usando as entradas de azimute, altitude, claridade e escala fornecidas.
- ST_Histogram - Retorna um conjunto de registros que resumem um raster ou distribuição de dados de cobertura raster intervalos bin separados. O número de bins é auto calculado.
- ST_MakeEmptyRaster - Interpolates a gridded surface based on an input set of 3-d points, using the X- and Y-values to position the points on the grid and the Z-value of the points as the surface elevation.
- ST_Intersection - Retorna uma raster ou conjunto de pares de valores de pixeis de geometria representando a porção dividida de dois rasters ou a interseção geométrica de uma vetorização do raster e uma geometria.
- ST_Intersects - Retorna verdade se o raster rastA intersectar espacialmente com o raster rastB.
- ST_IsEmpty - Retorna verdadeiro se o raster estiver vazio (largura = 0 e altura = 0). Senão, retorna falso.
- ST_MakeEmptyCoverage - Cover georeferenced area with a grid of empty raster tiles.
- ST_MakeEmptyRaster - Retorna um raster vazio (sem bandas) das dimensões dadas (width & height), o X e Y do superior esquerdo, tamanho de pixel e rotação (scalex, scaley, skewx & skewy) e sistema de referência (srid). Se um raster passar, retorna um novo raster com o mesmo tamanho, alinhamento e SRID. Se o srid é deixado de fora, a referência espacial se torna desconhecida (0).
- Funções retorno de mapa algébrico embutido - Versão função retorno - Retorna um raster de uma banda dado um ou mais rasters de entrada, os índices e uma função retorno de um usuário específico.
- ST_MapAlgebraExpr - Versão de banda raster 1: Cria uma nova banda raster formada pela aplicação de ma operação algébrica válida do PostgreSQL na banda raster de entrada de um tipo de pixel fornecido. A banda 1 é assumida se nenhuma banda for especificada.
- ST_MapAlgebraExpr - Versão de banda raster 2: Cria uma banda raster nova formada pela aplicação de uma operação algébrica válida PostgreSQL nas duas bandas raster de entrada e do tipo de pixel fornecido. A banda 1 de cada raster é assumida se nenhum número de bandas for especificado. O raster resultante será alinhado (escala, inclinação e cantos de pixel) na grade definida pelo primeiro raster e tem sua extensão definida pelo parâmetro "extenttype". O valores para "extenttype" pode ser: INTERSEÇÃO, UNIÃO, PRIMEIRO, SEGUNDO.
- ST_MapAlgebraFct - Versão de banda raster 1: Cria uma nova banda raster formada pela aplicação de uma função válida do PostgreSQL na banda raster de entrada de um tipo de pixel fornecido. A banda 1 é assumida se nenhuma banda for especificada.
- ST_MapAlgebraFct - Versão de banda 2 - Cria uma nova banda raster um formada pela aplicação de uma função PostgreSQL na 2 entrada de bandas raster e do tipo de pixel fornecido. A banda 1 é assumida se nenhuma banda for especificada. Tipo de extensão torna-se INTERSEÇÃO se não especificada.
- ST_MapAlgebraFctNgb - Versão 1-banda: o vizinho mais próximo no mapa algébrico usando a função de usuário definido PostgreSQL. Retorna um raster cujos valores são o resultado de uma função usuário PLPGSQL envolvendo uma vizinhança de valores da banda raster de entrada.
- ST_MapAlgebraExpr - Versão expressão - Retorna um raster de uma banda dado um ou mais rasters de entrada, índices de banda e uma ou mais expressões SQL de usuários específicos.

- ST_MemSize - Retorna a quantidade de espaço (em bytes) que o raster pega.
- ST_MetaData - Retorna metadados básicos sobre um objeto raster como um tanho pixel, rotação (skew), esquerda superior, inferior etc.
- ST_MinConvexHull - Retorna a geometria de casco convexo do raster excluindo os pixeis SEM DADOS.
- ST_NearestValue - Retorna o valor não-NODATA mais próximo de um dado pixel de banda especificado por uma colunax e linhay ou um ponto geométrico expressado no mesmo sistema de coordenada referência do raster.
- ST_Neighborhood - Retorna um arranjo de precisão 2-D dobrada dos valores não-NODATA em torno da banda de pixel especificada ou por uma colunaX e linhaY ou um ponto geométrico expressado no mesmo sistema de coordenada de referência especial como o raster.
- ST_NotSameAlignmentReason - Retorna a declaração de texto se os rasters estiverem alinhados e se não tiverem, uma razão do porquê.
- ST_NumBands - Retorna o número de bandas no objeto raster.
- ST_Overlaps - Retorna verdade se o raster rastA e rastB se intersectam, mas um deles não contém o outro completamente.
- ST_PixelAsCentroid - Retorna o centroide (ponto) da área representada por um pixel.
- ST_PixelAsCentroids - Retorna o centroide (ponto geométrico) para cada pixel de uma banda raster junto com o valor, as coordenas raster X e Y de cada pixel. O ponto é o centroide da área representada por um pixel.
- ST_PixelAsPoint - Retorna um ponto geométrico do canto superior esquerdo do pixel.
- ST_PixelAsPoints - Retorna um ponto geométrico para cada pixel de uma banda raster junto com o valor, as coordenas raster X e Y de cada pixel. As coordenadas do ponto são do ponto esquerdo superior do pixel.
- ST_PixelAsPolygon - Retorna o polígono que limita o pixel para uma linha e coluna específicas.
- ST_PixelAsPolygons - Retorna o polígono que limita cada pixel de uma banda raster ao longo do valor, as coordenadas raster X e Y de cada pixel.
- ST_PixelHeight - Retorna a altura do pixel em unidades geométricas do sistema de referência espacial.
- ST_PixelOfValue - Obtenha as coordenadas colunax, linhay do pixel cujos valores são iguais ao valor de pesquisa.
- ST_PixelWidth - Retorna a largura do pixel em unidades geométricas do sistema de referência espacial.
- ST_Polygon - Retorna um multipolígono formado pela união de pixeis que têm um valor que não é um valor sem dados. Se um número de banda for especificado, usa-se 1.
- ST_Quantile - Calcula quantiles para um raster ou cobertura de tabela raster no contexto da amostra ou população. Assim, um valor poderia ser examinado para estar na porcentagem 25%, 50%, 75% do raster.
- ST_RastFromHexWKB - Return a raster value from a Hex representation of Well-Known Binary (WKB) raster.
- ST_RastFromWKB - Return a raster value from a Well-Known Binary (WKB) raster.
- ST_RasterToWorldCoord - Retorna o canto superior esquerdo do raster como X e Y geométricos (longitude e latitude) dada a coluna e linha. Coluna e linha começam em 1.
- ST_RasterToWorldCoordX - Retorna a coordenada geométrica X superior esquerda de um raster, coluna ou linha. A numeração das colunas e linhas começam no 1.
- ST_RasterToWorldCoordY - Retorna a coordenada geométrica Y superior esquerda de um raster, coluna e linha. A numeração das colunas e linhas começam no 1.
- ST_Reclass - Cria um novo raster composto por tipos de banda reclassificados do original. A nband pode ser alterada. Se nenhuma nband for especificada, usa-se a 1. Todas as outras bandas são retornadas inalteradas. Use caso: converta uma banda 16BUI para 8BUI e então adiante para uma renderização mais simples como formatos visíveis.

- ST_Resample - Resample um raster usando um algorítimo específico, novas dimensões, um canto aleatório da grade e um conjunto de rasters georreferenciando atributos definidos ou emprestados de outro raster.
- ST_Rescale - Resample a raster by adjusting only its scale (or pixel size). New pixel values are computed using the NearestNeighbor (english or american spelling), Bilinear, Cubic, CubicSpline, Lanczos, Max or Min resampling algorithm. Default is NearestNeighbor.
- ST_Resize - Redimensiona largura/altura novas para um raster
- ST_Reskew - Resample um raster ajustando somente sua inclinação (ou tamanho de pixel). Novos valores de pixel são calculados usando o algorítimo NearestNeighbor (english or american spelling), Bilinear, Cubic, CubicSpline ou Lanczos. O padrão é NearestNeighbor.
- ST_Rotation - Retorna a rotação do raster em radianos.
- ST_Roughness - Retorna um raster com a "robustez" calculada de um DEM.
- ST_SRID - Retorna o identificador de referência espacial como definido na tabela spatial_ref_sys.
- ST_SameAlignment - Retorna verdade se os rasters têm a mesma inclinação, escala, referência espacial, e deslocamento (pixeis podem ser colocados na mesma grade sem cortar eles) e falso se eles não notificarem problemas detalhados.
- ST_ScaleX - Retorna o componente X da largura do pixel em unidades do sistema de referência coordenadas.
- ST_ScaleY - Retorna o componente Y da altura do pixel em unidades do sistema de referência coordenadas.
- ST_SetBandIndex - Update the external band number of an out-db band
- ST_SetBandIsNoData - Coloca a bandeira isnodata da banda como VERDADE.
- ST_SetBandNoDataValue - Coloca o valor da banda que não representa nenhum dado. A banda 1 é assumida se nenhuma banda for especificada. Para marcar uma banda como tendo nenhum valor nodata, coloca ele = NULL.
- ST_SetBandPath - Update the external path and band number of an out-db band
- ST_SetGeoReference - Coloque os parâmetros Georeference 6 em uma única chamada. Os números deverão ser separadospor espaço branco. Aceita entrar no formato GDAL ou ESRI. O padrão é GDAL.
- ST_SetSkew - Returns a geometry with the same X/Y coordinates as the input geometry, and values from the raster copied into the M dimension using the requested resample algorithm.
- ST_SetRotation - Põe a rotação do raster em radianos.
- ST_SetSRID - Coloca o SRID de um raster em um srid inteiro específico definido na tabela spatial_ref_sys.
- ST_SetScale - Coloca os tamanhos X e Y dos pixeis em unidades do sistema referencial de coordenadas. Número unidades/pixel largura/altura.
- ST_SetSkew - Coloca as georreferências X e Y distorcidas (ou parâmetro de rotação). Se somente um passar, coloca o X e o Y no mesmo valor.
- ST_SetUpperLeft - Sets the value of the upper left corner of the pixel of the raster to projected X and Y coordinates.
- ST_SetValue - Retorna o raster modificado resultante do valor de uma banda em uma dada colunax, linhay pixel ou os pixeis que intersectam uma geometria específica. Os números de banda começam no 1 e são assumidos como 1 se não estiverem especificados.
- ST_SetValues - Retorna o raster modificado resultante dos valores de uma dada banda.
- ST_SetSkew - Returns a geometry with the same X/Y coordinates as the input geometry, and values from the raster copied into the Z dimension using the requested resample algorithm.
- ST_SkewX - Retorna o desvio X georreferência (ou parâmetro e rotação).
- ST_SkewY - Retorna o desvio Y georreferência (ou parâmetro e rotação).

- ST_Slope - Retorna o declive (em graus) de uma banda raster de elevação. Útil para analisar terrenos.
- ST_SnapToGrid - Resample um raster encaixando-o em uma grade. Novos valores de pixel são calculados usando o algorítimo NearestNeighbor (english or american spelling), Bilinear, Cubic, CubicSpline ou Lanczos. O padrão é NearestNeighbor.
- ST_Summary - Retorna um texto resumo dos conteúdos do raster.
- ST_SummaryStats - Retorna as estatísticas resumidas consistindo de count, sum, mean, stddev, min, max para uma dada banda raster de um raster ou cobertura raster. A banda 1 é assumida se nenhuma banda for especificada.
- ST_SummaryStatsAgg - Agregado. Retorna as estatísticas resumidas consistindo de count, sum, mean, stddev, min, max para uma dada banda raster de um conjunto de rasters. A banda 1 é assumida se nenhuma banda for especificada.
- ST_TPI - Retorna um raster com o índice de posição topográfico calculado.
- ST_TRI - Retorna um raster com o índice de aspereza do terreno calculado.
- ST_Tile - Retorna um conjunto de rasters resultante de uma divisão do raster de entrada baseado nas dimensões desejadas nos rasters de saída.
- ST_Touches - Retorna verdade se o raster rastA e rastB têm pelo menos um ponto em comum, mas seus interiores não se intersectarem.
- ST_Transform - Reprojeta um raster em um sistema de referência espacial conhecido para outro usando um algorítimo resampling especificado. As opções são NearestNeighbor, Bilinear, Cubic, CubicSpline, Lanczos com o padrão sendo NearestNeighbor.
- ST_Union - Retorna a união de um conjunto de tiles raster em um único raster composto de 1 ou mais bandas.
- ST_UpperLeftX - Retorna a coordenada X superior esquerda na ref. espacial projetada.
- ST_UpperLeftY - Retorna a coordenada Y superior esquerda na ref. espacial projetada.
- ST_Value - Retorna o valor da banda dada com a colunax, linhay pixel ou em um ponto específico. Os números de banda começam em 1 e assumem-se 1 se não especificados. Se exclude_nodata_value for falso, então todos os pixeis, inclusive os nodata, são considerados para intersectar e retornar valor. Se exclude_nodata_value não passar então lê dos metadados do raster.
- ST_ValueCount - Retorna o conjunto de registros contendo uma banda pixel de valor e conta do número de pixeis em uma dada banda de um raster (ou uma cobertura raster) que tem um dado conjunto de valores. Usa-se a banda 1 se nenhuma for especificada. Por padrão pixeis de valor nodata não são contados. Todos os outros valores no pixel são saída e os valores de pixeis são arredondados para o inteiro mais próximo.
- ST_Width - Retorna a largura do raster em pixeis.
- ST_Within - Retorna verdade se nenhum ponto do raster rastA estiver no exterior do raster rastB e pelo menos um ponto do interior do rastA estiver no interior do rastB.
- ST_WorldToRasterCoord - Retorna o canto superior esquerdo como coluna e linha dados os X e Y geométricos (longitude e latitude) ou um ponto expressado na coordenada do sistema de referência espacial do raster.
- ST_WorldToRasterCoordX - Retorna a coluna no raster do ponto (pt) ou uma coordenada X e Y (xw, yw) representada no sistema de referência espacial mundial de raster.
- ST_WorldToRasterCoordY - Retorna a linha no raster do ponto (pt) ou uma coordenada X e Y (xw, yw) representada no sistema de referência espacial global de raster.
- UpdateRasterSRID - Altera o SRID de todos os rasters na coluna e tabela do usuário especificado.
- ST_PixelOfValue - Retorna o número de bandas no objeto raster.

12.6 PostGIS Geometry / Geography / Raster Dump Functions

The functions given below are PostGIS functions that take as input or return as output a set of or single geometry_dump or geomval data type object.

- ST_DumpAsPolygons - Retorna um conjunto de linhas geomval (geom,val), de uma dada banda raster. Se nenhum número de banda for especificado, o número de banda torna-se 1.
- ST_Intersection - Retorna uma raster ou conjunto de pares de valores de pixeis de geometria representando a porção dividida de dois rasters ou a interseção geométrica de uma vetorização do raster e uma geometria.

12.7 PostGIS Box Functions

The functions given below are PostGIS functions that take as input or return as output the box* family of PostGIS spatial types. The box family of types consists of box2d, and box3d

- Box2D - Returns a BOX2D representing the 2D extent of a geometry.
- Box3D - Returns a BOX3D representing the 3D extent of a geometry.
- Caixa3D - Retorna a representação da caixa 3d da caixa encerrada do raster.
- ST_3DExtent - Aggregate function that returns the 3D bounding box of geometries.
- ST_3DMakeBox - Creates a BOX3D defined by two 3D point geometries.
- ST_AsMVTGeom - Transforms a geometry into the coordinate space of a MVT tile.
- ST_AsTWKB - Retorna a geometria como TWKB, também conhecido como "Tiny Well-Known Binary"
- ST_Box2dFromGeoHash - Retorna uma CAIXA2D de uma string GeoHash.
- ST_ClipByBox2D - Computes the portion of a geometry falling within a rectangle.
- ST_EstimatedExtent - Returns the estimated extent of a spatial table.
- ST_Expand - Returns a bounding box expanded from another bounding box or a geometry.
- ST_Extent - Aggregate function that returns the bounding box of geometries.
- ST_MakeBox2D - Creates a BOX2D defined by two 2D point geometries.
- ST_XMax - Returns the X maxima of a 2D or 3D bounding box or a geometry.
- ST_XMin - Returns the X minima of a 2D or 3D bounding box or a geometry.
- ST_YMax - Returns the Y maxima of a 2D or 3D bounding box or a geometry.
- ST_YMin - Returns the Y minima of a 2D or 3D bounding box or a geometry.
- ST_ZMax - Returns the Z maxima of a 2D or 3D bounding box or a geometry.
- ST_ZMin - Returns the Z minima of a 2D or 3D bounding box or a geometry.
- RemoveUnusedPrimitives - Removes topology primitives which not needed to define existing TopoGeometry objects.
- ValidateTopology - Returns a set of validatetopology_returntype objects detailing issues with topology.
- ~(box2df,box2df) - Returns TRUE if a 2D float precision bounding box (BOX2DF) contains another 2D float precision bounding box (BOX2DF).
- ~(box2df,geometry) - Returns TRUE if a 2D float precision bounding box (BOX2DF) contains a geometry's 2D bonding box.

- ~(geometry,box2df) - Returns TRUE if a geometry's 2D bonding box contains a 2D float precision bounding box (GIDX).
- @(box2df,box2df) - Returns TRUE if a 2D float precision bounding box (BOX2DF) is contained into another 2D float precision bounding box.
- @(box2df,geometry) - Returns TRUE if a 2D float precision bounding box (BOX2DF) is contained into a geometry's 2D bounding box.
- @(geometry,box2df) - Returns TRUE if a geometry's 2D bounding box is contained into a 2D float precision bounding box (BOX2DF).
- &&(box2df,box2df) - Returns TRUE if two 2D float precision bounding boxes (BOX2DF) intersect each other.
- &&(box2df,geometry) - Returns TRUE if a 2D float precision bounding box (BOX2DF) intersects a geometry's (cached) 2D bounding box.
- &&(geometry,box2df) - Returns TRUE if a geometry's (cached) 2D bounding box intersects a 2D float precision bounding box (BOX2DF).

12.8 PostGIS Functions that support 3D

The functions given below are PostGIS functions that do not throw away the Z-Index.

- AddGeometryColumn - Remove uma coluna geometria de uma spatial table.
- Box3D - Returns a BOX3D representing the 3D extent of a geometry.
- DropGeometryColumn - Remove uma coluna geometria de uma spatial table.
- GeometryType - Retorna o tipo de geometria de valor ST_Geometry.
- ST_3DArea - Computa a área de geometrias de superfície 3D. Irá retornar 0 para sólidos.
- ST_3DClosestPoint - Retorna o ponto 3 dimensional em g1 que é o mais próximo de g2. Este é o primeiro ponto da linha mais curta em três dimensões.
- ST_3DConvexHull - Computa o eixo mediano aproximado de uma geometria territorial.
- ST_3DDFullyWithin - Tests if two 3D geometries are entirely within a given 3D distance
- ST_3DDWithin - Tests if two 3D geometries are within a given 3D distance
- ST_3DDifference - Representar diferença 3D
- ST_3DDistance - Para tipo geometria, retorna a menor distância cartesiana 3-dimensional (baseado no sistema de referência espacial) entre duas geometrias em unidades projetadas.
- ST_3DExtent - Aggregate function that returns the 3D bounding box of geometries.
- ST_3DIntersection - Representar intersecção 3D
- ST_3DIntersects - Tests if two geometries spatially intersect in 3D - only for points, linestrings, polygons, polyhedral surface (area)
- ST_3DLength - Retorna o centro geométrico de uma geometria.
- ST_3DLineInterpolatePoint - Returns a point interpolated along a 3D line at a fractional location.
- ST_3DLongestLine - Retorna a linha 3-dimensional mais longa entre duas geometrias
- ST_3DMaxDistance - Para tipo de geometria retorna a maior distância 3-dimensional cartesiana (baseada na referência espacial) entre duas geometrias em unidade projetadas.
- ST_3DPerimeter - Retorna o centro geométrico de uma geometria.

- ST_3DShortestLine - Retorna a menor linha 3-dimensional entre duas geometrias
- ST_3DUnion - Perform 3D union.
- ST_AddMeasure - Interpolates measures along a linear geometry.
- ST_AddPoint - Adicione um ponto para uma LineString.
- ST_Affine - Apply a 3D affine transformation to a geometry.
- ST_ApproximateMedialAxis - Computa o eixo mediano aproximado de uma geometria territorial.
- ST_AsBinary - Return the OGC/ISO Well-Known Binary (WKB) representation of the geometry/geography without SRID meta data.
- ST_AsEWKB - Return the Extended Well-Known Binary (EWKB) representation of the geometry with SRID meta data.
- ST_AsEWKT - Retorna a representação de texto bem conhecida (WKT) da geometria com os meta dados SRID.
- ST_AsGML - Retorna a geometria como uma versão GML com 2 ou 3 elementos.
- ST_AsGeoJSON - Return a geometry as a GeoJSON element.
- ST_AsHEXEWKB - Retorna uma geometria no formato HEXEWKB (como texto) usando little-endian (NDR) ou big-endian (XDR) encoding.
- ST_AsKML - Retorna a geometria como uma versão GML com 2 ou 3 elementos.
- ST_AsX3D - Retorna uma geometria em X3D nó xml formato do elemento: ISO-IEC-19776-1.2-X3DEncodings-XML
- ST_Boundary - Retorna o encerramento da borda combinatória dessa geometria.
- ST_BoundingDiagonal - Retorna a diagonal da geometria fornecida da caixa limitada.
- ST_CPAWithin - Tests if the closest point of approach of two trajectories is within the specified distance.
- ST_ChaikinSmoothing - Returns a smoothed version of a geometry, using the Chaikin algorithm
- ST_ClosestPointOfApproach - Returns a measure at the closest point of approach of two trajectories.
- ST_Collect - Creates a GeometryCollection or Multi* geometry from a set of geometries.
- ST_ConstrainedDelaunayTriangles - Return a constrained Delaunay triangulation around the given input geometry.
- ST_ConvexHull - Computes the convex hull of a geometry.
- ST_CoordDim - Retorna a dimensão da coordenada do valor ST_Geometry.
- ST_CurveToLine - Converts a geometry containing curves to a linear geometry.
- ST_DelaunayTriangles - Returns the Delaunay triangulation of the vertices of a geometry.
- ST_Difference - Computes a geometry representing the part of geometry A that does not intersect geometry B.
- ST_DistanceCPA - Returns the distance between the closest point of approach of two trajectories.
- ST_Dump - Returns a set of geometry_dump rows for the components of a geometry.
- ST_DumpPoints - Retorna um texto resumo dos conteúdos da geometria.
- ST_DumpRings - Returns a set of geometry_dump rows for the exterior and interior rings of a Polygon.
- ST_DumpSegments - Retorna um texto resumo dos conteúdos da geometria.
- ST_EndPoint - Retorna o número de pontos em um valor ST_LineString ou ST_CircularString.
- ST_ExteriorRing - Retorna o número de anéis interiores de um polígono.
- ST_Extrude - Extrude uma superfície a um volume relacionado

- ST_FlipCoordinates - Returns a version of a geometry with X and Y axis flipped.
- ST_Force2D - Força a geometria para o modo de 2 dimensões.
- ST_ForceCurve - Converte para cima uma geometria para seu tipo curvo, se aplicável.
- ST_ForceLHR - Orientação força LHR
- ST_ForcePolygonCCW - Orients all exterior rings counter-clockwise and all interior rings clockwise.
- ST_ForcePolygonCW - Orients all exterior rings clockwise and all interior rings counter-clockwise.
- ST_ForceRHR - Força a orientação dos vértices em um polígono a seguir a regra da mão direita.
- ST_ForceSFS - Força as geometrias a utilizarem os tipos disponíveis na especificação SFS 1.1.
- ST_Force_3D - Força a geometria para um modo XYZ. Este é um apelido para a função ST_Force_3DZ.
- ST_Force_3DZ - Força as geometrias para o modo XYZ.
- ST_Force_4D - Força as geometrias para o modo XYZM.
- ST_Force_Collection - Converte a geometria para um GEOMETRYCOLLECTION.
- ST_GeomFromEWKB - Retorna um valor ST_Geometry especifico da representação binária estendida bem conhecida (EWKB).
- ST_GeomFromEWKT - Retorna um valor ST_Geometry especifico da representação de texto estendida bem conhecida (EWKT).
- ST_GeomFromGML - Utiliza como entrada uma representação GML de geometria e como saída um objeto de geometria PostGIS
- ST_GeomFromGeoJSON - Utiliza como entrada uma representação geojson de uma geometria e como saída um objeto de geometria PostGIS
- ST_GeomFromKML - Utiliza como entrada uma representação KML de geometria e como saída um objeto de geometria PostGIS
- ST_GeometricMedian - Retorna a mediana de um MultiPonto.
- ST_GeometryN - Retorna o tipo de geometria de valor ST_Geometry.
- ST_GeometryType - Retorna o tipo de geometria de valor ST_Geometry.
- ST_HasArc - Tests if a geometry contains a circular arc
- ST_InteriorRingN - Retorna o número de anéis interiores de um polígono.
- ST_InterpolatePoint - Retorna o valor da dimensão de medida da geometria no ponto fechado para o ponto fornecido.
- ST_Intersection - Computes a geometry representing the shared portion of geometries A and B.
- ST_IsClosed - Retorna VERDADEIRO se os pontos de começo e fim da LINESTRING são coincidentes. Para superfície poliédrica está fechada (volumétrica).
- ST_IsCollection - Retorna verdadeiro se essa geometria é uma coleção vazia, polígono, ponto etc.
- ST_IsPlanar - Verifique se a superfície é ou não planar
- ST_IsPolygonCCW - Tests if Polygons have exterior rings oriented counter-clockwise and interior rings oriented clockwise.
- ST_IsPolygonCW - Tests if Polygons have exterior rings oriented clockwise and interior rings oriented counter-clockwise.
- ST_IsSimple - Retorna (VERDADEIRA) se essa geometria não tem nenhum ponto irregular, como auto intersecção ou tangenciação.
- ST_IsSolid - teste se a geometria é um sólido. Nenhuma verificação de validade é representada.
- ST_IsValidTrajectory - Tests if the geometry is a valid trajectory.

- ST_Length_Spheroid - Retorna o centro geométrico de uma geometria.
- ST_LineFromMultiPoint - Cria uma linestring de um multiponto geométrico.
- ST_LineInterpolatePoint - Returns a point interpolated along a line at a fractional location.
- ST_LineInterpolatePoints - Returns points interpolated along a line at a fractional interval.
- ST_LineSubstring - Returns the part of a line between two fractional locations.
- ST_LineToCurve - Converts a linear geometry to a curved geometry.
- ST_LocateBetweenElevations - Returns the portions of a geometry that lie in an elevation (Z) range.
- ST_M - Returns the M coordinate of a Point.
- ST_MakeLine - Cria uma Linestring de ponto, multiponto ou linha das geometrias.
- ST_MakePoint - Creates a 2D, 3DZ or 4D Point.
- ST_MakePolygon - Creates a Polygon from a shell and optional list of holes.
- ST_MakeSolid - Molde a geometria para um sólido. Nenhuma verificação é apresentada. Para obter um sólido válido, a geometria de entrada deve ser uma superfície poliédrica fechada ou um TIN fechado.
- ST_MakeValid - Attempts to make an invalid geometry valid without losing vertices.
- ST_MemSize - Retorna o tipo de geometria de valor ST_Geometry.
- ST_MemUnion - Aggregate function which unions geometries in a memory-efficent but slower way
- ST_NDims - Retorna a dimensão da coordenada do valor ST_Geometry.
- ST_NPoints - Retorna o número de pontos (vértices) em uma geometria.
- ST_NRings - Retorna o número de anéis interiores de um polígono.
- ST_Node - Nodes a collection of lines.
- ST_NumGeometries - Retorna o número de pontos em uma geometria. Funciona para todas as geometrias.
- ST_NumPatches - Retorna o número de faces em uma superfícies poliédrica. Retornará nulo para geometrias não poliédricas.
- ST_Orientation - Determine orientação da superfície
- ST_PatchN - Retorna o tipo de geometria de valor ST_Geometry.
- ST_PointFromWKB - Faz uma geometria a partir de um WKB com o SRID dado
- ST_PointN - Retorna o número de pontos em um valor ST_LineString ou ST_CircularString.
- ST_PointOnSurface - Computes a point guaranteed to lie in a polygon, or on a geometry.
- ST_Points - Retorna uma multilinestring contendo todas as coordenadas de uma geometria.
- ST_Polygon - Creates a Polygon from a LineString with a specified SRID.
- ST_RemovePoint - Remove a point from a linestring.
- ST_RemoveRepeatedPoints - Returns a version of a geometry with duplicate points removed.
- ST_Reverse - Retorna a geometria com a ordem dos vértices revertida.
- ST_Rotate - Rotates a geometry about an origin point.
- ST_RotateX - Rotates a geometry about the X axis.
- ST_RotateY - Rotates a geometry about the Y axis.

- ST_RotateZ - Rotates a geometry about the Z axis.
- ST_Scale - Scales a geometry by given factors.
- ST_Scroll - Change start point of a closed LineString.
- ST_SetPoint - Substitui ponto de uma linestring com um dado ponto.
- ST_ShiftLongitude - Shifts the longitude coordinates of a geometry between -180..180 and 0..360.
- ST_SnapToGrid - Rompe todos os pontos da geometria de entrada para uma rede regular.
- ST_StartPoint - Returns the first point of a LineString.
- ST_StraightSkeleton - Calcule um esqueleto em linha reta de uma geometria
- ST_SwapOrdinates - Retorna uma versão da geometria dada com os valores ordenados dados trocados.
- ST_SymDifference - Computes a geometry representing the portions of geometries A and B that do not intersect.
- ST_Tesselate - Representa superfície tesselação de um polígono ou superfície poliédrica e retorna como uma TIN ou coleção de TINS
- ST_TransScale - Translates and scales a geometry by given offsets and factors.
- ST_Translate - Translates a geometry by given offsets.
- ST_UnaryUnion - Computes the union of the components of a single geometry.
- ST_Union - Computes a geometry representing the point-set union of the input geometries.
- ST_Volume - Computa o volume de um sólido 3D. Se aplicado a geometrias com superfícies (mesmo fechadas), irão retornar 0.
- ST_WrapX - Envolve uma geometria em torno de um valor X.
- ST_X - Returns the X coordinate of a Point.
- ST_XMax - Returns the X maxima of a 2D or 3D bounding box or a geometry.
- ST_XMin - Returns the X minima of a 2D or 3D bounding box or a geometry.
- ST_Y - Returns the Y coordinate of a Point.
- ST_YMax - Returns the Y maxima of a 2D or 3D bounding box or a geometry.
- ST_YMin - Returns the Y minima of a 2D or 3D bounding box or a geometry.
- ST_Z - Returns the Z coordinate of a Point.
- ST_ZMax - Returns the Z maxima of a 2D or 3D bounding box or a geometry.
- ST_ZMin - Returns the Z minima of a 2D or 3D bounding box or a geometry.
- ST_Zmflag - Retorna a dimensão da coordenada do valor ST_Geometry.
- TG_Equals - Retorna verdade se duas topogeometrias forem compostas da mesma topologia primitiva
- TG_Intersects - Retorna verdade se algum par de primitivos das duas topologias se intersectar.
- UpdateGeometrySRID - Updates the SRID of all features in a geometry column, and the table metadata.
- geometry_overlaps_nd - Retorna VERDADE se a caixa limitadora n-D de A intersecta a caixa limitadora n-D de B.
- overlaps_nd_geometry_gidx - Returns TRUE if a geometry's (cached) n-D bounding box intersects a n-D float precision bounding box (GIDX).

- overlaps_nd_gidx_geometry - Returns TRUE if a n-D float precision bounding box (GIDX) intersects a geometry's (cached) n-D bounding box.
- overlaps_nd_gidx_gidx - Returns TRUE if two n-D float precision bounding boxes (GIDX) intersect each other.
- postgis_sfcgal_full_version - Returns the full version of SFCGAL in use including CGAL and Boost versions
- postgis_sfcgal_version - retorna a versão do SFCGAL em uso

12.9 PostGIS Curved Geometry Support Functions

The functions given below are PostGIS functions that can use CIRCULARSTRING, CURVEPOLYGON, and other curved geometry types

- AddGeometryColumn - Remove uma coluna geometria de uma spatial table.
- Box2D - Returns a BOX2D representing the 2D extent of a geometry.
- Box3D - Returns a BOX3D representing the 3D extent of a geometry.
- DropGeometryColumn - Remove uma coluna geometria de uma spatial table.
- Tipo de geometria - Retorna o tipo de geometria de valor ST_Geometry.
- PostGIS_AddBBox - Add bounding box to the geometry.
- PostGIS_DropBBox - Drop the bounding box cache from the geometry.
- PostGIS_HasBBox - Returns TRUE if the bbox of this geometry is cached, FALSE otherwise.
- ST_3DExtent - Aggregate function that returns the 3D bounding box of geometries.
- ST_Affine - Apply a 3D affine transformation to a geometry.
- ST_AsBinary - Return the OGC/ISO Well-Known Binary (WKB) representation of the geometry/geography without SRID meta data.
- ST_AsEWKB - Return the Extended Well-Known Binary (EWKB) representation of the geometry with SRID meta data.
- ST_AsEWKT - Retorna a representação de texto bem conhecida (WKT) da geometria com os meta dados SRID.
- ST_AsHEXEWKB - Retorna uma geometria no formato HEXEWKB (como texto) usando little-endian (NDR) ou big-endian (XDR) encoding.
- ST_AsSVG - Returns SVG path data for a geometry.
- ST_AsText - Retorna a representação de texto bem conhecida (WKT) da geometria/geografia sem os meta dados do SRID.
- ST_ClusterDBSCAN - Window function that returns a cluster id for each input geometry using the DBSCAN algorithm.
- ST_ClusterWithin - Aggregate function that clusters input geometries by separation distance.
- ST_ClusterWithinWin - Window function that returns a cluster id for each input geometry, clustering using separation distance.
- ST_GeomCollFromText - Creates a GeometryCollection or Multi* geometry from a set of geometries.
- ST_CoordDim - Retorna a dimensão da coordenada do valor ST_Geometry.
- ST_CurveToLine - Converts a geometry containing curves to a linear geometry.
- ST_Distance - Retorna a linha 3-dimensional mais longa entre duas geometrias
- ST_Dump - Returns a set of geometry_dump rows for the components of a geometry.
- ST_NumPoints - Retorna um texto resumo dos conteúdos da geometria.

- ST_EndPoint - Retorna o número de pontos em um valor ST_LineString ou ST_CircularString.
- ST_EstimatedExtent - Returns the estimated extent of a spatial table.
- ST_FlipCoordinates - Returns a version of a geometry with X and Y axis flipped.
- ST_Force2D - Força a geometria para o modo de 2 dimensões.
- ST_ForceCurve - Converte para cima uma geometria para seu tipo curvo, se aplicável.
- ST_ForceSFS - Força as geometrias a utilizarem os tipos disponíveis na especificação SFS 1.1.
- ST_Force3D - Força a geometria para um modo XYZ. Este é um apelido para a função ST_Force_3DZ.
- ST_Force3DM - Força as geometrias para o modo XYM.
- ST_Force3DZ - Força as geometrias para o modo XYZ.
- ST_Force4D - Força as geometrias para o modo XYZM.
- ST_ForceCollection - Converte a geometria para um GEOMETRYCOLLECTION.
- ST_GeoHash - Retorna uma representação GeoHash da geometria.
- ST_GeogFromWKB - Cria uma ocasião geografia de uma geometria binária bem conhecida (WKB) ou binário estendido bem conhecido (EWKB).
- ST_GeomFromEWKB - Retorna um valor ST_Geometry especifico da representação binária estendida bem conhecida (EWKB).
- ST_GeomFromEWKT - Retorna um valor ST_Geometry especifico da representação de texto estendida bem conhecida (EWKT).
- ST_GeomFromText - Retorna um valor ST_Geometry específico da representação de texto bem conhecida (WKT).
- ST_GeomFromWKB - Criar uma geometria exemplo de um representação bem conhecida de geometria binária (WKB) e SRID opcional.
- ST_GeometryN - Retorna o tipo de geometria de valor ST_Geometry.
- = - Returns TRUE if the coordinates and coordinate order geometry/geography A are the same as the coordinates and coordinate order of geometry/geography B.
- &<| - Retorna VERDADE se a caixa limitadora de A sobrepõe ou está abaixo de B.
- ST_HasArc - Tests if a geometry contains a circular arc
- ST_Intersects - Tests if two geometries intersect (they have at least one point in common)
- ST_IsClosed - Retorna VERDADEIRO se os pontos de começo e fim da LINESTRING são coincidentes. Para superfície poliédrica está fechada (volumétrica).
- ST_IsCollection - Retorna verdadeiro se essa geometria é uma coleção vazia, polígono, ponto etc.
- ST_IsEmpty - Tests if a geometry is empty.
- ST_LineToCurve - Converts a linear geometry to a curved geometry.
- ST_MemSize - Retorna o tipo de geometria de valor ST_Geometry.
- ST_NPoints - Retorna o número de pontos (vértices) em uma geometria.
- ST_NRings - Retorna o número de anéis interiores de um polígono.
- ST_PointFromWKB - Faz uma geometria a partir de um WKB com o SRID dado
- ST_PointN - Retorna o número de pontos em um valor ST_LineString ou ST_CircularString.
- ST_Points - Retorna uma multilinestring contendo todas as coordenadas de uma geometria.

- ST_Rotate - Rotates a geometry about an origin point.
- ST_RotateZ - Rotates a geometry about the Z axis.
- ST_SRID - Returns the spatial reference identifier for a geometry.
- ST_Scale - Scales a geometry by given factors.
- ST_SetSRID - Set the SRID on a geometry.
- ST_StartPoint - Returns the first point of a LineString.
- ST_Summary - Retorna um texto resumo dos conteúdos da geometria.
- ST_SwapOrdinates - Retorna uma versão da geometria dada com os valores ordenados dados trocados.
- ST_TransScale - Translates and scales a geometry by given offsets and factors.
- ST_Transform - Return a new geometry with coordinates transformed to a different spatial reference system.
- ST_Translate - Translates a geometry by given offsets.
- ST_XMax - Returns the X maxima of a 2D or 3D bounding box or a geometry.
- ST_XMin - Returns the X minima of a 2D or 3D bounding box or a geometry.
- ST_YMax - Returns the Y maxima of a 2D or 3D bounding box or a geometry.
- ST_YMin - Returns the Y minima of a 2D or 3D bounding box or a geometry.
- ST_ZMax - Returns the Z maxima of a 2D or 3D bounding box or a geometry.
- ST_ZMin - Returns the Z minima of a 2D or 3D bounding box or a geometry.
- ST_Zmflag - Retorna a dimensão da coordenada do valor ST_Geometry.
- UpdateGeometrySRID - Updates the SRID of all features in a geometry column, and the table metadata.
- ~(box2df,box2df) - Returns TRUE if a 2D float precision bounding box (BOX2DF) contains another 2D float precision bounding box (BOX2DF).
- ~(box2df,geometry) - Returns TRUE if a 2D float precision bounding box (BOX2DF) contains a geometry's 2D bonding box.
- ~(geometry,box2df) - Returns TRUE if a geometry's 2D bonding box contains a 2D float precision bounding box (GIDX).
- && - Retorna VERDADE se a caixa limitadora 2D de A intersecta a caixa limitadora 2D de B.
- &&& - Retorna VERDADE se a caixa limitadora n-D de A intersecta a caixa limitadora n-D de B.
- @(box2df,box2df) - Returns TRUE if a 2D float precision bounding box (BOX2DF) is contained into another 2D float precision bounding box.
- @(box2df,geometry) - Returns TRUE if a 2D float precision bounding box (BOX2DF) is contained into a geometry's 2D bounding box.
- @(geometry,box2df) - Returns TRUE if a geometry's 2D bounding box is contained into a 2D float precision bounding box (BOX2DF).
- &&(box2df,box2df) - Returns TRUE if two 2D float precision bounding boxes (BOX2DF) intersect each other.
- &&(box2df,geometry) - Returns TRUE if a 2D float precision bounding box (BOX2DF) intersects a geometry's (cached) 2D bounding box.
- &&(geometry,box2df) - Returns TRUE if a geometry's (cached) 2D bounding box intersects a 2D float precision bounding box (BOX2DF).
- &&&(geometry,gidx) - Returns TRUE if a geometry's (cached) n-D bounding box intersects a n-D float precision bounding box (GIDX).
- &&&(gidx,geometry) - Returns TRUE if a n-D float precision bounding box (GIDX) intersects a geometry's (cached) n-D bounding box.
- &&&(gidx,gidx) - Returns TRUE if two n-D float precision bounding boxes (GIDX) intersect each other.

12.10 PostGIS Polyhedral Surface Support Functions

The functions given below are PostGIS functions that can use POLYHEDRALSURFACE, POLYHEDRALSURFACEM geometries

- AddGeometryColumn - Remove uma coluna geometria de uma spatial table.
- Box2D - Returns a BOX2D representing the 2D extent of a geometry.
- Box3D - Returns a BOX3D representing the 3D extent of a geometry.
- DropGeometryColumn - Remove uma coluna geometria de uma spatial table.
- Tipo de geometria - Retorna o tipo de geometria de valor ST_Geometry.
- PostGIS_AddBBox - Add bounding box to the geometry.
- PostGIS_DropBBox - Drop the bounding box cache from the geometry.
- PostGIS_HasBBox - Returns TRUE if the bbox of this geometry is cached, FALSE otherwise.
- ST_3DExtent - Aggregate function that returns the 3D bounding box of geometries.
- ST_Affine - Apply a 3D affine transformation to a geometry.
- ST_AsBinary - Return the OGC/ISO Well-Known Binary (WKB) representation of the geometry/geography without SRID meta data.
- ST_AsEWKB - Return the Extended Well-Known Binary (EWKB) representation of the geometry with SRID meta data.
- ST_AsEWKT - Retorna a representação de texto bem conhecida (WKT) da geometria com os meta dados SRID.
- ST_AsHEXEWKB - Retorna uma geometria no formato HEXEWKB (como texto) usando little-endian (NDR) ou big-endian (XDR) encoding.
- ST_AsSVG - Returns SVG path data for a geometry.
- ST_AsText - Retorna a representação de texto bem conhecida (WKT) da geometria/geografia sem os meta dados do SRID.
- ST_ClusterDBSCAN - Window function that returns a cluster id for each input geometry using the DBSCAN algorithm.
- ST_ClusterWithin - Aggregate function that clusters input geometries by separation distance.
- ST_ClusterWithinWin - Window function that returns a cluster id for each input geometry, clustering using separation distance.
- ST_GeomCollFromText - Creates a GeometryCollection or Multi* geometry from a set of geometries.
- ST_CoordDim - Retorna a dimensão da coordenada do valor ST_Geometry.
- ST_CurveToLine - Converts a geometry containing curves to a linear geometry.
- ST_Distance - Retorna a linha 3-dimensional mais longa entre duas geometrias
- ST_Dump - Returns a set of geometry_dump rows for the components of a geometry.
- ST_NumPoints - Retorna um texto resumo dos conteúdos da geometria.
- ST_EndPoint - Retorna o número de pontos em um valor ST_LineString ou ST_CircularString.
- ST_EstimatedExtent - Returns the estimated extent of a spatial table.
- ST_FlipCoordinates - Returns a version of a geometry with X and Y axis flipped.
- ST_Force2D - Força a geometria para o modo de 2 dimensões.
- ST_ForceCurve - Converte para cima uma geometria para seu tipo curvo, se aplicável.

- ST_ForceSFS - Força as geometrias a utilizarem os tipos disponíveis na especificação SFS 1.1.
- ST_Force3D - Força a geometria para um modo XYZ. Este é um apelido para a função ST_Force_3DZ.
- ST_Force3DM - Força as geometrias para o modo XYM.
- ST_Force3DZ - Força as geometrias para o modo XYZ.
- ST_Force4D - Força as geometrias para o modo XYZM.
- ST_ForceCollection - Converte a geometria para um GEOMETRYCOLLECTION.
- ST_GeoHash - Retorna uma representação GeoHash da geometria.
- ST_GeogFromWKB - Cria uma ocasião geografia de uma geometria binária bem conhecida (WKB) ou binário estendido bem conhecido (EWKB).
- ST_GeomFromEWKB - Retorna um valor ST_Geometry especifico da representação binária estendida bem conhecida (EWKB).
- ST_GeomFromEWKT - Retorna um valor ST_Geometry especifico da representação de texto estendida bem conhecida (EWKT).
- ST_GeomFromText - Retorna um valor ST_Geometry específico da representação de texto bem conhecida (WKT).
- ST_GeomFromWKB - Criar uma geometria exemplo de um representação bem conhecida de geometria binária (WKB) e SRID opcional.
- ST_GeometryN - Retorna o tipo de geometria de valor ST_Geometry.
- = - Returns TRUE if the coordinates and coordinate order geometry/geography A are the same as the coordinates and coordinate order of geometry/geography B.
- &<| - Retorna VERDADE se a caixa limitadora de A sobrepõe ou está abaixo de B.
- ST_HasArc - Tests if a geometry contains a circular arc
- ST_Intersects - Tests if two geometries intersect (they have at least one point in common)
- ST_IsClosed - Retorna VERDADEIRO se os pontos de começo e fim da LINESTRING são coincidentes. Para superfície poliédrica está fechada (volumétrica).
- ST_IsCollection - Retorna verdadeiro se essa geometria é uma coleção vazia, polígono, ponto etc.
- ST_IsEmpty - Tests if a geometry is empty.
- ST_LineToCurve - Converts a linear geometry to a curved geometry.
- ST_MemSize - Retorna o tipo de geometria de valor ST_Geometry.
- ST_NPoints - Retorna o número de pontos (vértices) em uma geometria.
- ST_NRings - Retorna o número de anéis interiores de um polígono.
- ST_PointFromWKB - Faz uma geometria a partir de um WKB com o SRID dado
- ST_PointN - Retorna o número de pontos em um valor ST_LineString ou ST_CircularString.
- ST_Points - Retorna uma multilinestring contendo todas as coordenadas de uma geometria.
- ST_Rotate - Rotates a geometry about an origin point.
- ST_RotateZ - Rotates a geometry about the Z axis.
- ST_SRID - Returns the spatial reference identifier for a geometry.
- ST_Scale - Scales a geometry by given factors.
- ST_SetSRID - Set the SRID on a geometry.

- ST_StartPoint - Returns the first point of a LineString.
- ST_Summary - Retorna um texto resumo dos conteúdos da geometria.
- ST_SwapOrdinates - Retorna uma versão da geometria dada com os valores ordenados dados trocados.
- ST_TransScale - Translates and scales a geometry by given offsets and factors.
- ST_Transform - Return a new geometry with coordinates transformed to a different spatial reference system.
- ST_Translate - Translates a geometry by given offsets.
- ST_XMax - Returns the X maxima of a 2D or 3D bounding box or a geometry.
- ST_XMin - Returns the X minima of a 2D or 3D bounding box or a geometry.
- ST_YMax - Returns the Y maxima of a 2D or 3D bounding box or a geometry.
- ST_YMin - Returns the Y minima of a 2D or 3D bounding box or a geometry.
- ST_ZMax - Returns the Z maxima of a 2D or 3D bounding box or a geometry.
- ST_ZMin - Returns the Z minima of a 2D or 3D bounding box or a geometry.
- ST_Zmflag - Retorna a dimensão da coordenada do valor ST_Geometry.
- UpdateGeometrySRID - Updates the SRID of all features in a geometry column, and the table metadata.
- ~(box2df,box2df) - Returns TRUE if a 2D float precision bounding box (BOX2DF) contains another 2D float precision bounding box (BOX2DF).
- ~(box2df,geometry) - Returns TRUE if a 2D float precision bounding box (BOX2DF) contains a geometry's 2D bonding box.
- ~(geometry,box2df) - Returns TRUE if a geometry's 2D bonding box contains a 2D float precision bounding box (GIDX).
- && - Retorna VERDADE se a caixa limitadora 2D de A intersecta a caixa limitadora 2D de B.
- &&& - Retorna VERDADE se a caixa limitadora n-D de A intersecta a caixa limitadora n-D de B.
- @(box2df,box2df) - Returns TRUE if a 2D float precision bounding box (BOX2DF) is contained into another 2D float precision bounding box.
- @(box2df,geometry) - Returns TRUE if a 2D float precision bounding box (BOX2DF) is contained into a geometry's 2D bounding box.
- @(geometry,box2df) - Returns TRUE if a geometry's 2D bounding box is contained into a 2D float precision bounding box (BOX2DF).
- &&(box2df,box2df) - Returns TRUE if two 2D float precision bounding boxes (BOX2DF) intersect each other.
- &&(box2df,geometry) - Returns TRUE if a 2D float precision bounding box (BOX2DF) intersects a geometry's (cached) 2D bounding box.
- &&(geometry,box2df) - Returns TRUE if a geometry's (cached) 2D bounding box intersects a 2D float precision bounding box (BOX2DF).
- &&&(geometry,gidx) - Returns TRUE if a geometry's (cached) n-D bounding box intersects a n-D float precision bounding box (GIDX).
- &&&(gidx,geometry) - Returns TRUE if a n-D float precision bounding box (GIDX) intersects a geometry's (cached) n-D bounding box.
- &&&(gidx,gidx) - Returns TRUE if two n-D float precision bounding boxes (GIDX) intersect each other.

12.11 PostGIS Function Support Matrix

Below is an alphabetical listing of spatial specific functions in PostGIS and the kinds of spatial types they work with or OGC/SQL compliance they try to conform to.

- A ✓ means the function works with the type or subtype natively.
- A  means it works but with a transform cast built-in using cast to geometry, transform to a "best srid" spatial ref and then cast back. Results may not be as expected for large areas or areas at poles and may accumulate floating point junk.
- A  means the function works with the type because of a auto-cast to another such as to box3d rather than direct type support.
- A  means the function only available if PostGIS compiled with SFCGAL support.
- A  means the function support is provided by SFCGAL if PostGIS compiled with SFCGAL support, otherwise GEOS/built-in support.
- geom - Basic 2D geometry support (x,y).
- geog - Basic 2D geography support (x,y).
- 2.5D - basic 2D geometries in 3 D/4D space (has Z or M coord).
- PS - Polyhedral surfaces
- T - Triangles and Triangulated Irregular Network surfaces (TIN)

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
Box2D	✓			✓		✓	✓
Box3D	✓		✓	✓		✓	✓
Tipo de geometria			✓	✓		✓	✓
PostGIS_AddBBox	✓			✓			
PostGIS_DropBBox	✓			✓			
PostGIS_HasBBox	✓			✓			
ST_3DArea							
ST_3DClosestPoint	✓		✓			✓	
ST_3DConvexHull							
ST_3DDFullyWithi	✓		✓			✓	
ST_3DDWithin	✓		✓		✓	✓	
ST_3DDifference							
ST_3DDistance	✓		✓		✓	✓	
ST_3DExtent	✓		✓	✓		✓	✓

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_3DIntersection							
ST_3DIntersects	✓		✓		✓	✓	✓
ST_3DLength	✓		✓		✓		
ST_LineInterpolatePoint			✓				
ST_3DLongestLine	✓		✓			✓	
ST_3DMakeBox	✓						
ST_3DMaxDistance	✓		✓			✓	
ST_3DPerímetro	✓		✓		✓		
ST_3DShortestLine	✓		✓			✓	
ST_3DUnion							
ST_AddMeasure			✓				
ST_AddPoint	✓		✓				
ST_Affine	✓		✓	✓		✓	✓
ST_AlphaShape							
ST_Angle	✓						
ST_ApproximateMedialAxis							
ST_Area	✓	✓			✓	✓	
ST_AsBinary	✓	✓	✓	✓	✓	✓	✓
ST_AsEWKB	✓		✓	✓		✓	✓
ST_AsEWKT	✓	✓	✓	✓		✓	✓
ST_AsEncodedPoly	✓						
ST_AsFlatGeobuf							
ST_AsGML	✓	✓	✓		✓	✓	✓
ST_AsGeoJSON	✓	✓	✓				
ST_AsGeobuf							
ST_AsHEXEWKB	✓		✓	✓			
ST_AsKML	✓	✓	✓				
ST_AsLatLonText	✓						
ST_AsMARC21	✓						
ST_AsMVT							
ST_AsMVTGeom	✓						
ST_AsSVG	✓	✓		✓			

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_AsTWKB	✓						
ST_AsText	✓	✓		✓	✓		
ST_AsX3D	✓		✓			✓	✓
ST_Azimuth	✓	✓					
ST_BdMPolyFromText							
ST_BdPolyFromText							
ST_Boundary			✓		✓		
ST_BoundingDiagonal			✓				
ST_Box2dFromGeoHash							
ST_Buffer	✓				✓		
ST_BuildArea	✓						
ST_CPAWithin	✓		✓				
ST_Centroid	✓	✓			✓		
ST_ChaikinSmooth	✓		✓				
ST_ClipByBox2D	✓						
ST_ClosestPoint	✓	✓					
ST_ClosestPointOfA	✓	oach	✓				
ST_ClusterDBSCA	✓			✓			
ST_ClusterIntersect	✓						
ST_ClusterIntersect	✓	Vin					
ST_ClusterKMeans	✓						
ST_ClusterWithin	✓			✓			
ST_ClusterWithinW	✓			✓			
ST_GeomCollFrom	✓		✓	✓			
ST_CollectionExtra	✓						
ST_CollectionHom	✓	ize					
ST_ConcaveHull	✓						
ST_ConstrainedDe	yTriangles						
ST_Contains	✓				✓		
ST_ContainsProper	✓						
ST_ConvexHull	✓		✓		✓		
ST_CoordDim			✓	✓	✓	✓	✓
ST_CoverageInvalic	✓	es					
ST_CoverageSimpli	✓						

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_CoverageUnion	✓						
ST_CoveredBy	✓	✓					
ST_Covers	✓	✓					
ST_Crosses	✓				✓		
ST_CurveToLine	✓		✓	✓	✓		
ST_DFullyWithin	✓						
ST_DWithin	✓	✓					
ST_DelaunayTriang	✓		✓				✓
ST_Difference	✓		✓		✓		
ST_Dimension					✓	✓	✓
ST_Disjoint	✓				✓		
ST_Distance	✓	✓		✓	✓		
ST_DistanceCPA	✓		✓				
ST_DistanceSphere	✓						
ST_DistanceSphero	✓						
ST_Dump			✓	✓		✓	✓
ST_NumPoints			✓	✓		✓	✓
ST_NRings	✓		✓				
ST_NumPoints			✓				✓
ST_EndPoint			✓	✓	✓		
ST_Envelope					✓		
ST_Equals	✓				✓		
ST_EstimatedExten	✓			✓			
ST_Expand	✓					✓	✓
ST_Extent	✓					✓	✓
ST_ExteriorRing	✓		✓		✓		
ST_Extrude							
ST_FilterByM	✓						
ST_FlipCoordinates	✓		✓	✓		✓	✓
ST_Force2D	✓		✓	✓		✓	
ST_ForceCurve	✓		✓	✓			
ST_ForceLHR							
ST_ForcePolygonC	✓		✓				

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_ForcePolygonC	✓		✓				
ST_ForceRHR	✓		✓			✓	
ST_ForceSFS	✓		✓	✓		✓	✓
ST_Force3D	✓		✓	✓		✓	
ST_Force3DM	✓			✓			
ST_Force3DZ	✓		✓	✓		✓	
ST_Force4D	✓		✓	✓			
ST_ForceCollection	✓		✓	✓		✓	
ST_FrechetDistance	✓						
ST_FromFlatGeobuf							
ST_FromFlatGeobufToTable							
ST_GMLToSQL	✓				✓		
ST_GeneratePoints	✓						
ST_GeoHash	✓			✓			
ST_GeogFromText		✓					
ST_GeogFromWKB		✓		✓			
ST_GeographyFromText		✓					
ST_GeomCollFrom	✓				✓		
ST_GeomFromEWKB			✓	✓		✓	✓
ST_GeomFromEWKT			✓	✓		✓	✓
ST_GeomFromGM	✓		✓			✓	✓
ST_GeomFromGeoHash							
ST_GeomFromGeo	✓ N		✓				
ST_GeomFromKML			✓				
ST_GeomFromMA	✓ 1						
ST_GeomFromTWKB							
ST_GeomFromText	✓			✓	✓		
ST_GeomFromWK	✓			✓	✓		
ST_GeometricMedi	✓		✓				
ST_GeometryFrom	✓				✓		
ST_GeometryN			✓	✓	✓	✓	✓
ST_GeometryType			✓		✓	✓	
|>>	✓						
<<|	✓						
~	✓						

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
@	✓						
=	✓	✓		✓		✓	
<<	✓						
|&>	✓						
&<|	✓			✓		✓	
&<	✓						
&>	✓						
>>	✓						
~=	✓					✓	
ST_HasArc			✓	✓			
ST_HausdorffDistan	✓						
ST_Hexagon	✓						
ST_HexagonGrid							
ST_InteriorRingN			✓		✓		
ST_InterpolatePoint	✓		✓				
ST_Intersection	✓		✓		✓		
ST_Intersects	✓	✓		✓	✓		✓
ST_InverseTransfor ✓ peline							
ST_IsClosed			✓	✓	✓	✓	
ST_IsCollection			✓	✓			
ST_IsEmpty				✓	✓		
ST_IsPlanar							
ST_IsPolygonCCW	✓		✓				
ST_IsPolygonCW	✓		✓				
ST_IsRing					✓		
ST_IsSimple			✓		✓		
ST_IsSolid							
ST_IsValid	✓				✓		
ST_IsValidDetail	✓						
ST_IsValidReason	✓						
ST_IsValidTrajector	✓		✓				
ST_LargestEmptyC ✓							
ST_Length	✓	✓			✓		

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_Length2D	✓						
ST_LengthSpheroid	✓		✓				
ST_Letters	✓						
ST_LineCrossingDirection	✓						
ST_LineExtend	✓						
ST_LineFromEncodedPolyline							
ST_LineFromMultiPoint			✓				
ST_LineFromText	✓				✓		
ST_LineFromWKB	✓				✓		
ST_LineInterpolatePoint	✓	✓	✓				
ST_LineInterpolatePoints	✓	✓	✓				
ST_LineLocatePoint	✓	✓					
ST_LineMerge	✓						
ST_LineSubstring	✓	✓	✓				
ST_LineToCurve	✓		✓	✓			
ST_LinestringFromWKB	✓				✓		
ST_LocateAlong					✓		
ST_LocateBetween					✓		
ST_LocateBetweenElevations			✓				
ST_LongestLine	✓						
ST_M			✓		✓		
ST_MLineFromText	✓				✓		
ST_MPointFromText	✓				✓		
ST_MPolyFromText	✓				✓		
ST_MakeBox2D	✓						
ST_MakeEnvelope	✓						
ST_MakeLine	✓		✓				
ST_MakePoint			✓				
ST_MakePointM	✓						
ST_MakePolygon	✓		✓				
ST_MakeSolid							
ST_MakeValid	✓		✓				
ST_MaxDistance	✓						
ST_MaximumInscribedCircle	✓						

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_MemSize			✓	✓		✓	✓
ST_MemUnion	✓		✓				
ST_MinimumBoun	✓ Circle						
ST_MinimumBoun	✓ Radius						
ST_MinimumClear	✓						
ST_MinimumClear	✓ Line						
ST_MinkowskiSun							
ST_Multi	✓						
ST_NDims			✓				
ST_NPoints			✓	✓		✓	
ST_NRings			✓	✓			
ST_Node	✓		✓				
ST_Normalize	✓						
ST_NumGeometries			✓		✓	✓	✓
ST_NumInteriorRing							
ST_NumInteriorRings					✓		
ST_NumPatches			✓		✓	✓	
ST_NumPoints					✓		
ST_OffsetCurve	✓						
ST_OptimalAlphaS							
ST_OrderingEquals	✓				✓		
ST_Orientation							
ST_OrientedEnvelo	✓						
ST_Overlaps	✓				✓		
ST_PatchN			✓		✓	✓	
ST_Perimeter	✓	✓			✓		
ST_Perimeter2D	✓						
ST_Point					✓		
ST_PointFromGeoHash							
ST_PointFromText	✓				✓		
ST_PointFromWKB	✓		✓	✓	✓		
ST_PointInsideCirc	✓						
ST_Point	✓						

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_PointN			✓	✓	✓		
ST_PointOnSurface	✓		✓		✓		
ST_Point	✓						
ST_Point	✓						
ST_Points			✓	✓			
ST_Polygon	✓		✓		✓		
ST_PolygonFromTe	✓				✓		
ST_Polygonize	✓						
ST_Project	✓	✓					
ST_QuantizeCoordi	✓ s						
ST_ReducePrecisio	✓						
ST_Relate	✓				✓		
ST_RelateMatch							
ST_RemovePoint	✓		✓				
ST_RemoveRepeate	✓ ints		✓			✓	
ST_Reverse	✓		✓			✓	
ST_Rotate	✓		✓	✓		✓	✓
ST_RotateX	✓		✓			✓	✓
ST_RotateY	✓		✓			✓	✓
ST_RotateZ	✓		✓	✓		✓	✓
ST_SRID	✓			✓	✓		
ST_Scale	✓		✓	✓		✓	✓
ST_Scroll	✓		✓				
ST_Segmentize	✓	✓					
ST_SetEffectiveAre	✓						
ST_SetPoint	✓		✓				
ST_SetSRID	✓			✓			
ST_SharedPaths	✓						
ST_ShiftLongitude	✓		✓			✓	✓
ST_ShortestLine	✓	✓					
ST_Simplify	✓						
ST_SimplifyPolygo	✓ ll						
ST_SimplifyPreserv	✓ pology						
ST_SimplifyVW	✓						

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_Snap	✓						
ST_SnapToGrid	✓		✓				
ST_Split	✓						
ST_Square	✓						
ST_SquareGrid							
ST_StartPoint			✓	✓	✓		
ST_StraightSkeleto							
ST_Subdivide	✓						
ST_Summary	✓	✓		✓		✓	✓
ST_SwapOrdinates	✓		✓	✓		✓	✓
ST_SymDifference	✓		✓		✓		
ST_Tesselate							
ST_MakeEnvelope	✓						
ST_Touches	✓				✓		
ST_TransScale	✓		✓	✓			
ST_Transform	✓			✓	✓	✓	
ST_TransformPipel	✓						
ST_Translate	✓		✓	✓			
ST_TriangulatePoly	✓						
ST_UnaryUnion	✓		✓				
ST_Union	✓		✓		✓		
ST_Volume							
ST_VoronoiLines	✓						
ST_VoronoiPolygor	✓						
ST_WKBToSQL	✓				✓		
ST_WKTToSQL					✓		
ST_Within	✓				✓		
ST_WrapX	✓		✓				
ST_X			✓		✓		
ST_XMax			✓	✓			
ST_XMin			✓	✓			
ST_Y			✓		✓		
ST_YMax			✓	✓			

Function	geom	geog	2.5D	Curves	SQL MM	PS	T
ST_YMin	✓		✓	✓			
ST_Z			✓		✓		
ST_ZMax	✓		✓	✓			
ST_ZMin	✓		✓	✓			
ST_Zmflag			✓	✓			
~(box2df,box2df)	✓			✓		✓	
~(box2df,geometry)	✓			✓		✓	
~(geometry,box2df)	✓			✓		✓	
<#>	✓						
<<#>>	✓						
<<->>	✓						
|=|	✓						
<->	✓	✓					
&&	✓	✓		✓		✓	
&&&	✓		✓	✓		✓	✓
@(box2df,box2df)	✓			✓		✓	
@(box2df,geometry	✓			✓		✓	
@(geometry,box2df	✓			✓		✓	
&&(box2df,box2df)	✓			✓		✓	
&&(box2df,geomet	✓			✓		✓	
&&(geometry,box2	✓			✓		✓	
&&&(geometry,gid	✓		✓	✓		✓	✓
&&&(gidx,geometr	✓		✓	✓		✓	✓
&&&(gidx,gidx)			✓	✓		✓	✓
postgis.backend							
postgis.enable_outdb_rasters							
postgis.gdal_datapath							
postgis.gdal_enabled_drivers							
postgis.gdal_datapath							
postgis_sfcgal_version							
postgis_sfcgal_version							
postgis_srs							
postgis_srs_all							
postgis_srs_codes							
postgis_srs_search	✓						

12.12 New, Enhanced or changed PostGIS Functions

12.12.1 PostGIS Functions new or enhanced in 3.4

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 3.4

- PostGIS_GEOS_Compiled_Version - Availability: 3.4.0 Returns the version number of the GEOS library against which PostGIS was built.
- ST_ClusterIntersectingWin - Availability: 3.4.0 Window function that returns a cluster id for each input geometry, clustering input geometries into connected sets.
- ST_ClusterWithinWin - Availability: 3.4.0 Window function that returns a cluster id for each input geometry, clustering using separation distance.
- ST_CoverageInvalidEdges - Availability: 3.4.0 - requires GEOS >= 3.12.0 Window function that finds locations where polygons fail to form a valid coverage.
- ST_CoverageSimplify - Availability: 3.4.0 - requires GEOS >= 3.12.0 Window function that simplifies the edges of a polygonal coverage.
- ST_CoverageUnion - Availability: 3.4.0 - requires GEOS >= 3.8.0 Computes the union of a set of polygons forming a coverage by removing shared edges.
- ST_InverseTransformPipeline - Availability: 3.4.0 Return a new geometry with coordinates transformed to a different spatial reference system using the inverse of a defined coordinate transformation pipeline.
- ST_LargestEmptyCircle - Availability: 3.4.0. Computes the largest circle not overlapping a geometry.
- ST_LineExtend - Availability: 3.4.0 Returns a line with the last and first segments extended the specified distance(s).
- ST_TransformPipeline - Availability: 3.4.0 Return a new geometry with coordinates transformed to a different spatial reference system using a defined coordinate transformation pipeline.
- postgis_srs - Availability: 3.4.0 Return a metadata record for the requested authority and srid.
- postgis_srs_all - Availability: 3.4.0 Return metadata records for every spatial reference system in the underlying Proj database.
- postgis_srs_codes - Availability: 3.4.0 Return the list of SRS codes associated with the given authority.
- postgis_srs_search - Availability: 3.4.0 Return metadata records for projected coordinate systems that have areas of useage that fully contain the bounds parameter.

Functions enhanced in PostGIS 3.4

- PostGIS_Full_Version - Enhanced: 3.4.0 now includes extra PROJ configurations NETWORK_ENABLED, URL_ENDPOINT and DATABASE_PATH of proj.db location Reports full PostGIS version and build configuration infos.
- PostGIS_PROJ_Version - Enhanced: 3.4.0 now includes NETWORK_ENABLED, URL_ENDPOINT and DATABASE_PATH of proj.db location Returns the version number of the PROJ4 library.
- ST_AsSVG - Enhanced: 3.4.0 to support all curve types Returns SVG path data for a geometry.
- ST_ClosestPoint - Enhanced: 3.4.0 - Support for geography. Returns the 2D point on g1 that is closest to g2. This is the first point of the shortest line from one geometry to the other.
- ST_Project - Enhanced: 3.4.0 Allow geometry arguments and two-point form omitting azimuth. Returns a point projected from a start point by a distance and bearing (azimuth).
- ST_ShortestLine - Enhanced: 3.4.0 - support for geography. Retorna a menor linha 2-dimensional entre duas geometrias

Functions changed in PostGIS 3.4

- PostGIS_Extensions_Upgrade - Changed: 3.4.0 to add target_version argument. Packages and upgrades PostGIS extensions (e.g. postgis_raster, postgis_topology, postgis_sfcgal) to given or latest version.

12.12.2 PostGIS Functions new or enhanced in 3.3

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 3.3

- ST_3DConvexHull - Disponibilidade: 2.1.0 Computa o eixo mediano aproximado de uma geometria territorial.
- ST_3DUnion - Availability: 3.3.0 aggregate variant was added Perform 3D union.
- ST_AlphaShape - Availability: 3.3.0 - requires SFCGAL >= 1.4.1. Computes an Alpha-shape enclosing a geometry
- ST_AsMARC21 - Availability: 3.3.0 Returns geometry as a MARC21/XML record with a geographic datafield (034).
- ST_GeomFromMARC21 - Availability: 3.3.0, requires libxml2 2.6+ Takes MARC21/XML geographic data as input and returns a PostGIS geometry object.
- ST_Letters - Disponibilidade: 2.1.0 Returns the input letters rendered as geometry with a default start position at the origin and default text height of 100.
- ST_OptimalAlphaShape - Availability: 3.3.0 - requires SFCGAL >= 1.4.1. Computes an Alpha-shape enclosing a geometry using an "optimal" alpha value.
- ST_SimplifyPolygonHull - Availability: 3.3.0. Computes a simplifed topology-preserving outer or inner hull of a polygonal geometry.
- ST_TriangulatePolygon - Availability: 3.3.0. Computes the constrained Delaunay triangulation of polygons
- postgis_sfcgal_version - Disponibilidade: 2.1.0 Returns the full version of SFCGAL in use including CGAL and Boost versions

Functions enhanced in PostGIS 3.3

- ST_ConcaveHull - Enhanced: 3.3.0, GEOS native implementation enabled for GEOS 3.11+ Computes a possibly concave geometry that contains all input geometry vertices
- ST_LineMerge - Enhanced: 3.3.0 accept a directed parameter. Return the lines formed by sewing together a MultiLineString.

Functions changed in PostGIS 3.3

- PostGIS_Extensions_Upgrade - Changed: 3.3.0 support for upgrades from any PostGIS version. Does not work on all systems. Packages and upgrades PostGIS extensions (e.g. postgis_raster, postgis_topology, postgis_sfcgal) to given or latest version.

12.12.3 PostGIS Functions new or enhanced in 3.2

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 3.2

- ST_AsFlatGeobuf - Availability: 3.2.0 Return a FlatGeobuf representation of a set of rows.
- ST_FromFlatGeobuf - Availability: 3.2.0 Reads FlatGeobuf data.
- ST_FromFlatGeobufToTable - Availability: 3.2.0 Creates a table based on the structure of FlatGeobuf data.
- ST_NumPoints - Disponibilidade: 2.2.0 Retorna um texto resumo dos conteúdos da geometria.
- ST_Scroll - Availability: 3.2.0 Change start point of a closed LineString.
- postgis.gdal_datapath - Disponibilidade: 2.2.0 Uma opção de configuração booleana para ativar o acesso ao out-db raster bands.

Functions enhanced in PostGIS 3.2

- ST_ClusterKMeans - Enhanced: 3.2.0 Support for max_radius Window function that returns a cluster id for each input geometry using the K-means algorithm.
- ST_MakeValid - Enhanced: 3.2.0, added algorithm options, 'linework' and 'structure' which requires GEOS >= 3.10.0. Attempts to make an invalid geometry valid without losing vertices.
- ST_Point - Enhanced: 3.2.0 srid as an extra optional argument was added. Older installs require combining with ST_SetSRID to mark the srid on the geometry. Creates a Point with X, Y and SRID values.
- ST_Point - Enhanced: 3.2.0 srid as an extra optional argument was added. Older installs require combining with ST_SetSRID to mark the srid on the geometry. Creates a Point with X, Y, Z and SRID values.
- ST_Point - Enhanced: 3.2.0 srid as an extra optional argument was added. Older installs require combining with ST_SetSRID to mark the srid on the geometry. Creates a Point with X, Y, M and SRID values.
- ST_Point - Enhanced: 3.2.0 srid as an extra optional argument was added. Older installs require combining with ST_SetSRID to mark the srid on the geometry. Creates a Point with X, Y, Z, M and SRID values.
- ST_RemovePoint - Enhanced: 3.2.0 Remove a point from a linestring.
- ST_RemoveRepeatedPoints - Enhanced: 3.2.0 Returns a version of a geometry with duplicate points removed.
- ST_StartPoint - Enhanced: 3.2.0 returns a point for all geometries. Prior behavior returns NULLs if input was not a LineString. Returns the first point of a LineString.

Functions changed in PostGIS 3.2

- ST_Boundary - Changed: 3.2.0 support for TIN, does not use geos, does not linearize curves Retorna o encerramento da borda combinatória dessa geometria.

12.12.4 PostGIS Functions new or enhanced in 3.1

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 3.1

- ST_Hexagon - Disponibilidade: 2.1.0 Returns a single hexagon, using the provided edge size and cell coordinate within the hexagon grid space.
- ST_HexagonGrid - Disponibilidade: 2.1.0 Returns a set of hexagons and cell indices that completely cover the bounds of the geometry argument.
- ST_MaximumInscribedCircle - Availability: 3.1.0. Retorna o centro geométrico de uma geometria.
- ST_ReducePrecision - Availability: 3.1.0. Returns a valid geometry with points rounded to a grid tolerance.
- ST_Square - Disponibilidade: 2.1.0 Returns a single square, using the provided edge size and cell coordinate within the square grid space.
- ST_SquareGrid - Disponibilidade: 2.1.0 Returns a set of grid squares and cell indices that completely cover the bounds of the geometry argument.

Functions enhanced in PostGIS 3.1

- ST_AsEWKT - Enhanced: 3.1.0 support for optional precision parameter. Retorna a representação de texto bem conhecida (WKT) da geometria com os meta dados SRID.
- ST_ClusterKMeans - Enhanced: 3.1.0 Support for 3D geometries and weights Window function that returns a cluster id for each input geometry using the K-means algorithm.
- ST_Difference - Enhanced: 3.1.0 accept a gridSize parameter. Computes a geometry representing the part of geometry A that does not intersect geometry B.

- ST_Intersection - Enhanced: 3.1.0 accept a gridSize parameter Computes a geometry representing the shared portion of geometries A and B.
- ST_MakeEnvelope - Melhorias: 2.0.0 parâmetro opcional padrão srid adicionado. Creates a rectangular Polygon in Web Mercator (SRID:3857) using the XYZ tile system.
- ST_MakeValid - Enhanced: 3.1.0, added removal of Coordinates with NaN values. Attempts to make an invalid geometry valid without losing vertices.
- ST_Subdivide - Enhanced: 3.1.0 accept a gridSize parameter. Computes a rectilinear subdivision of a geometry.
- ST_SymDifference - Enhanced: 3.1.0 accept a gridSize parameter. Computes a geometry representing the portions of geometries A and B that do not intersect.
- ST_UnaryUnion - Enhanced: 3.1.0 accept a gridSize parameter. Computes the union of the components of a single geometry.
- ST_Union - Enhanced: 3.1.0 accept a gridSize parameter. Computes a geometry representing the point-set union of the input geometries.

Functions changed in PostGIS 3.1

- ST_Force3D - Changed: 3.1.0. Added support for supplying a non-zero Z value. Força a geometria para um modo XYZ. Este é um apelido para a função ST_Force_3DZ.
- ST_Force3DM - Changed: 3.1.0. Added support for supplying a non-zero M value. Força as geometrias para o modo XYM.
- ST_Force3DZ - Changed: 3.1.0. Added support for supplying a non-zero Z value. Força as geometrias para o modo XYZ.
- ST_Force4D - Changed: 3.1.0. Added support for supplying non-zero Z and M values. Força as geometrias para o modo XYZM.

12.12.5 PostGIS Functions new or enhanced in 3.0

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 3.0

- ST_ConstrainedDelaunayTriangles - Disponibilidade: 2.1.0 Return a constrained Delaunay triangulation around the given input geometry.
- ST_LineInterpolatePoint - Disponibilidade: 2.0.0 Returns a point interpolated along a 3D line at a fractional location.
- ST_MakeEnvelope - Disponibilidade: 2.1.0 Creates a rectangular Polygon in Web Mercator (SRID:3857) using the XYZ tile system.

Functions enhanced in PostGIS 3.0

- ST_AsMVT - Enhanced: 3.0 - added support for Feature ID. Aggregate function returning a MVT representation of a set of rows.
- ST_Contains - Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION Tests if every point of B lies in A, and their interiors have a point in common
- ST_ContainsProperly - Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION Tests if every point of B lies in the interior of A
- ST_CoveredBy - Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION Tests if every point of A lies in B
- ST_Covers - Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION Tests if every point of B lies in A
- ST_Crosses - Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION Tests if two geometries have some, but not all, interior points in common

- ST_CurveToLine - Enhanced: 3.0.0 implemented a minimum number of segments per linearized arc to prevent topological collapse. Converts a geometry containing curves to a linear geometry.
- ST_Disjoint - Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION Tests if two geometries have no points in common
- ST_Equals - Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION Tests if two geometries include the same set of points
- ST_GeneratePoints - Enhanced: 3.0.0, added seed parameter Generates random points contained in a Polygon or MultiPolygon.
- ST_GeomFromGeoJSON - Enhanced: 3.0.0 parsed geometry defaults to SRID=4326 if not specified otherwise. Utiliza como entrada uma representação geojson de uma geometria e como saída um objeto de geometria PostGIS
- ST_LocateBetween - Enhanced: 3.0.0 - added support for POLYGON, TIN, TRIANGLE. Returns the portions of a geometry that match a measure range.
- ST_LocateBetweenElevations - Enhanced: 3.0.0 - added support for POLYGON, TIN, TRIANGLE. Returns the portions of a geometry that lie in an elevation (Z) range.
- ST_Overlaps - Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION Tests if two geometries have the same dimension and intersect, but each has at least one point not in the other
- ST_Relate - Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION Tests if two geometries have a topological relationship matching an Intersection Matrix pattern, or computes their Intersection Matrix
- ST_Segmentize - Enhanced: 3.0.0 Segmentize geometry now produces equal-length subsegments Returns a modified geometry/geography having no segment longer than a given distance.
- ST_Touches - Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION Tests if two geometries have at least one point in common, but their interiors do not intersect
- ST_Within - Enhanced: 3.0.0 enabled support for GEOMETRYCOLLECTION Tests if every point of A lies in B, and their interiors have a point in common

Functions changed in PostGIS 3.0

- PostGIS_Extensions_Upgrade - Changed: 3.0.0 to repackage loose extensions and support postgis_raster. Packages and upgrades PostGIS extensions (e.g. postgis_raster, postgis_topology, postgis_sfcgal) to given or latest version.
- ST_3DDistance - Changed: 3.0.0 - SFCGAL version removed Para tipo geometria, retorna a menor distância cartesiana 3-dimensional (baseado no sistema de referência espacial) entre duas geometrias em unidades projetadas.
- ST_3DIntersects - Changed: 3.0.0 SFCGAL backend removed, GEOS backend supports TINs. Tests if two geometries spatially intersect in 3D - only for points, linestrings, polygons, polyhedral surface (area)
- ST_Area - Changed: 3.0.0 - does not depend on SFCGAL anymore. Retorna o centro geométrico de uma geometria.
- ST_AsGeoJSON - Changed: 3.0.0 support records as input Return a geometry as a GeoJSON element.
- ST_AsGeoJSON - Changed: 3.0.0 output SRID if not EPSG:4326. Return a geometry as a GeoJSON element.
- ST_AsKML - Changed: 3.0.0 - Removed the "versioned" variant signature Retorna a geometria como uma versão GML com 2 ou 3 elementos.
- ST_Distance - Changed: 3.0.0 - does not depend on SFCGAL anymore. Retorna a linha 3-dimensional mais longa entre duas geometrias
- ST_Intersection - Changed: 3.0.0 does not depend on SFCGAL. Computes a geometry representing the shared portion of geometries A and B.
- ST_Intersects - Changed: 3.0.0 SFCGAL version removed and native support for 2D TINS added. Tests if two geometries intersect (they have at least one point in common)
- ST_Union - Changed: 3.0.0 does not depend on SFCGAL. Computes a geometry representing the point-set union of the input geometries.

12.12.6 PostGIS Functions new or enhanced in 2.5

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 2.5

- PostGIS_Extensions_Upgrade - Availability: 2.5.0 Packages and upgrades PostGIS extensions (e.g. postgis_raster, postgis_topology, postgis_sfcgal) to given or latest version.
- ST_Angle - Availability: 2.5.0 Retorna a linha 3-dimensional mais longa entre duas geometrias
- ST_ChaikinSmoothing - Availability: 2.5.0 Returns a smoothed version of a geometry, using the Chaikin algorithm
- ST_FilterByM - Availability: 2.5.0 Removes vertices based on their M value
- ST_LineInterpolatePoints - Availability: 2.5.0 Returns points interpolated along a line at a fractional interval.
- ST_OrientedEnvelope - Availability: 2.5.0. Returns a minimum-area rectangle containing a geometry.
- ST_QuantizeCoordinates - Availability: 2.5.0 Sets least significant bits of coordinates to zero

Functions enhanced in PostGIS 2.5

- ST_AsMVT - Enhanced: 2.5.0 - added support parallel query. Aggregate function returning a MVT representation of a set of rows.
- ST_AsText - Enhanced: 2.5 - optional parameter precision introduced. Retorna a representação de texto bem conhecida (WKT) da geometria/geografia sem os meta dados do SRID.
- ST_Buffer - Enhanced: 2.5.0 - ST_Buffer geometry support was enhanced to allow for side buffering specification side=both|left|right. Computes a geometry covering all points within a given distance from a geometry.
- ST_GeomFromGeoJSON - Enhanced: 2.5.0 can now accept json and jsonb as inputs. Utiliza como entrada uma representação geojson de uma geometria e como saída um objeto de geometria PostGIS
- ST_GeometricMedian - Enhanced: 2.5.0 Added support for M as weight of points. Retorna a mediana de um MultiPonto.
- ST_Intersects - Enhanced: 2.5.0 Supports GEOMETRYCOLLECTION. Tests if two geometries intersect (they have at least one point in common)
- ST_OffsetCurve - Enhanced: 2.5 - added support for GEOMETRYCOLLECTION and MULTILINESTRING Returns an offset line at a given distance and side from an input line.
- ST_Scale - Enhanced: 2.5.0 support for scaling relative to a local origin (origin parameter) was introduced. Scales a geometry by given factors.
- ST_Split - Enhanced: 2.5.0 support for splitting a polygon by a multiline was introduced. Returns a collection of geometries created by splitting a geometry by another geometry.
- ST_Subdivide - Enhanced: 2.5.0 reuses existing points on polygon split, vertex count is lowered from 8 to 5. Computes a rectilinear subdivision of a geometry.

12.12.7 PostGIS Functions new or enhanced in 2.4

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 2.4

- ST_AsGeobuf - Availability: 2.4.0 Return a Geobuf representation of a set of rows.
- ST_AsMVT - Availability: 2.4.0 Aggregate function returning a MVT representation of a set of rows.
- ST_AsMVTGeom - Availability: 2.4.0 Transforms a geometry into the coordinate space of a MVT tile.

- ST_Centroid - Availability: 2.4.0 support for geography was introduced. Retorna o centro geométrico de uma geometria.
- ST_ForcePolygonCCW - Availability: 2.4.0 Orients all exterior rings counter-clockwise and all interior rings clockwise.
- ST_ForcePolygonCW - Availability: 2.4.0 Orients all exterior rings clockwise and all interior rings counter-clockwise.
- ST_FrechetDistance - Availability: 2.4.0 - requires GEOS >= 3.7.0 Retorna a menor linha 3-dimensional entre duas geometrias
- ST_IsPolygonCCW - Disponibilidade: 2.2.0 Tests if Polygons have exterior rings oriented counter-clockwise and interior rings oriented clockwise.
- ST_IsPolygonCW - Disponibilidade: 2.2.0 Tests if Polygons have exterior rings oriented clockwise and interior rings oriented counter-clockwise.

Functions enhanced in PostGIS 2.4

- ST_AsTWKB - Enhanced: 2.4.0 memory and speed improvements. Retorna a geometria como TWKB, também conhecido como "Tiny Well-Known Binary"
- ST_Covers - Enhanced: 2.4.0 Support for polygon in polygon and line in polygon added for geography type Tests if every point of B lies in A
- ST_CurveToLine - Enhanced: 2.4.0 added support for max-deviation and max-angle tolerance, and for symmetric output. Converts a geometry containing curves to a linear geometry.
- ST_Project - Enhanced: 2.4.0 Allow negative distance and non-normalized azimuth. Returns a point projected from a start point by a distance and bearing (azimuth).
- ST_Reverse - Enhanced: 2.4.0 support for curves was introduced. Retorna a geometria com a ordem dos vértices revertida.

Functions changed in PostGIS 2.4

- = - Changed: 2.4.0, in prior versions this was bounding box equality not a geometric equality. If you need bounding box equality, use instead. Returns TRUE if the coordinates and coordinate order geometry/geography A are the same as the coordinates and coordinate order of geometry/geography B.
- ST_Node - Changed: 2.4.0 this function uses GEOSNode internally instead of GEOSUnaryUnion. This may cause the resulting linestrings to have a different order and direction compared to PostGIS < 2.4. Nodes a collection of lines.

12.12.8 PostGIS Functions new or enhanced in 2.3

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 2.3

- &&&(geometry,gidx) - Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+. Returns TRUE if a geometry's (cached) n-D bounding box intersects a n-D float precision bounding box (GIDX).
- &&&(gidx,geometry) - Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+. Returns TRUE if a n-D float precision bounding box (GIDX) intersects a geometry's (cached) n-D bounding box.
- &&&(gidx,gidx) - Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+. Returns TRUE if two n-D float precision bounding boxes (GIDX) intersect each other.
- &&(box2df,box2df) - Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+. Returns TRUE if two 2D float precision bounding boxes (BOX2DF) intersect each other.
- &&(box2df,geometry) - Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+. Returns TRUE if a 2D float precision bounding box (BOX2DF) intersects a geometry's (cached) 2D bounding box.
- &&(geometry,box2df) - Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+. Returns TRUE if a geometry's (cached) 2D bounding box intersects a 2D float precision bounding box (BOX2DF).

- @(box2df,box2df) - Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+. Returns TRUE if a 2D float precision bounding box (BOX2DF) is contained into another 2D float precision bounding box.
- @(box2df,geometry) - Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+. Returns TRUE if a 2D float precision bounding box (BOX2DF) is contained into a geometry's 2D bounding box.
- @(geometry,box2df) - Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+. Returns TRUE if a geometry's 2D bounding box is contained into a 2D float precision bounding box (BOX2DF).
- ST_ClusterDBSCAN - Availability: 2.3.0 Window function that returns a cluster id for each input geometry using the DBSCAN algorithm.
- ST_ClusterKMeans - Availability: 2.3.0 Window function that returns a cluster id for each input geometry using the K-means algorithm.
- ST_GeneratePoints - Disponibilidade: 2.3.0 Generates random points contained in a Polygon or MultiPolygon.
- ST_GeometricMedian - Disponibilidade: 2.3.0 Retorna a mediana de um MultiPonto.
- ST_MakeLine - Disponibilidad: 2.0.0 - Suporte para elementos de entrada linestring foi introduzido Cria uma Linestring de ponto, multiponto ou linha das geometrias.
- ST_MinimumBoundingRadius - Disponibilidade - 2.3.0 Returns the center point and radius of the smallest circle that contains a geometry.
- ST_MinimumClearance - Disponibilidade: 2.3.0 Retorna a liquidação mínima de uma geometria, uma medida de uma robustez de uma geometria.
- ST_MinimumClearanceLine - Disponibilidade: 2.3.0 - requer GEOS >= 3.6.0 Retorna a LineString de dois pontos abrangendo a liquidação mínima de uma geometria.
- ST_Normalize - Disponibilidade: 2.3.0 Retorna a geometria na sua forma canônica.
- ST_Points - Disponibilidade: 2.3.0 Retorna uma multilinestring contendo todas as coordenadas de uma geometria.
- ST_VoronoiLines - Disponibilidade: 2.3.0 Returns the boundaries of the Voronoi diagram of the vertices of a geometry.
- ST_VoronoiPolygons - Disponibilidade: 2.3.0 Returns the cells of the Voronoi diagram of the vertices of a geometry.
- ST_WrapX - Availability: 2.3.0 requires GEOS Envolve uma geometria em torno de um valor X.
- ~(box2df,box2df) - Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+. Returns TRUE if a 2D float precision bounding box (BOX2DF) contains another 2D float precision bounding box (BOX2DF).
- ~(box2df,geometry) - Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+. Returns TRUE if a 2D float precision bounding box (BOX2DF) contains a geometry's 2D bonding box.
- ~(geometry,box2df) - Availability: 2.3.0 support for Block Range INdexes (BRIN) was introduced. Requires PostgreSQL 9.5+. Returns TRUE if a geometry's 2D bonding box contains a 2D float precision bounding box (GIDX).

Functions enhanced in PostGIS 2.3

- ST_Contains - Enhanced: 2.3.0 Enhancement to PIP short-circuit extended to support MultiPoints with few points. Prior versions only supported point in polygon. Tests if every point of B lies in A, and their interiors have a point in common
- ST_Covers - Enhanced: 2.3.0 Enhancement to PIP short-circuit for geometry extended to support MultiPoints with few points. Prior versions only supported point in polygon. Tests if every point of B lies in A
- ST_Expand - Enhanced: 2.3.0 support was added to expand a box by different amounts in different dimensions. Returns a bounding box expanded from another bounding box or a geometry.
- ST_Intersects - Enhanced: 2.3.0 Enhancement to PIP short-circuit extended to support MultiPoints with few points. Prior versions only supported point in polygon. Tests if two geometries intersect (they have at least one point in common)

- ST_Segmentize - Enhanced: 2.3.0 Segmentize geography now produces equal-length subsegments Returns a modified geometry/geography having no segment longer than a given distance.
- ST_Transform - Enhanced: 2.3.0 support for direct PROJ.4 text was introduced. Return a new geometry with coordinates transformed to a different spatial reference system.
- ST_Within - Enhanced: 2.3.0 Enhancement to PIP short-circuit for geometry extended to support MultiPoints with few points. Prior versions only supported point in polygon. Tests if every point of A lies in B, and their interiors have a point in common

Functions changed in PostGIS 2.3

- ST_PointN - Alterações: 2.3.0 : indexing negativo disponível (-1 é o último ponto) Retorna o número de pontos em um valor ST_LineString ou ST_CircularString.

12.12.9 PostGIS Functions new or enhanced in 2.2

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 2.2

- <<#>> - Disponibilidade: 2.2.0 -- KNN só está disponível para PostgreSQL 9.1+ Retorna a distância n-D entre as caixas limitadoras de A e B.
- <<->> - Disponibilidade: 2.2.0 -- KNN só está disponível para PostgreSQL 9.1+ Retorna a distância n-D entre as centroides das caixas limitadoras de A e B.
- ST_3DDifference - Disponibilidade: 2.2.0 Representar diferença 3D
- ST_3DUnion - Disponibilidade: 2.2.0 Perform 3D union.
- ST_ApproximateMedialAxis - Disponibilidade: 2.2.0 Computa o eixo mediano aproximado de uma geometria territorial.
- ST_AsEncodedPolyline - Disponibilidade: 2.2.0 Retorna uma Polilinha Encoded de uma geometria LineString.
- ST_AsTWKB - Disponibilidade: 2.2.0 Retorna a geometria como TWKB, também conhecido como "Tiny Well-Known Binary"
- ST_BoundingDiagonal - Disponibilidade: 2.2.0 Retorna a diagonal da geometria fornecida da caixa limitada.
- ST_CPAWithin - Availability: 2.2.0 Tests if the closest point of approach of two trajectories is within the specified distance.
- ST_ClipByBox2D - Availability: 2.2.0 Computes the portion of a geometry falling within a rectangle.
- ST_ClosestPointOfApproach - Availability: 2.2.0 Returns a measure at the closest point of approach of two trajectories.
- ST_ClusterIntersecting - Availability: 2.2.0 Aggregate function that clusters input geometries into connected sets.
- ST_ClusterWithin - Availability: 2.2.0 Aggregate function that clusters input geometries by separation distance.
- ST_DistanceCPA - Availability: 2.2.0 Returns the distance between the closest point of approach of two trajectories.
- ST_ForceCurve - Disponibilidade: 2.2.0 Converte para cima uma geometria para seu tipo curvo, se aplicável.
- ST_IsPlanar - Disponibilidade: 2.2.0: Isso foi documentado em 2.1.0, mas foi deixado de fora acidentalmente na 2.1. Verifique se a superfície é ou não planar
- ST_IsSolid - Disponibilidade: 2.2.0 teste se a geometria é um sólido. Nenhuma verificação de validade é representada.
- ST_IsValidTrajectory - Availability: 2.2.0 Tests if the geometry is a valid trajectory.
- ST_LineFromEncodedPolyline - Disponibilidade: 2.2.0 Cria uma LineString de uma Encoded Polyline.
- ST_MakeSolid - Disponibilidade: 2.2.0 Molde a geometria para um sólido. Nenhuma verificação é apresentada. Para obter um sólido válido, a geometria de entrada deve ser uma superfície poliédrica fechada ou um TIN fechado.

- ST_RemoveRepeatedPoints - Disponibilidade: 2.2.0 Returns a version of a geometry with duplicate points removed.
- ST_SetEffectiveArea - Disponibilidade: 2.2.0 Sets the effective area for each vertex, using the Visvalingam-Whyatt algorithm.
- ST_SimplifyVW - Disponibilidade: 2.2.0 Returns a simplified version of a geometry, using the Visvalingam-Whyatt algorithm
- ST_Subdivide - Availability: 2.2.0 Computes a rectilinear subdivision of a geometry.
- ST_SwapOrdinates - Disponibilidade: 2.2.0 Retorna uma versão da geometria dada com os valores ordenados dados trocados.
- ST_Volume - Disponibilidade: 2.2.0 Computa o volume de um sólido 3D. Se aplicado a geometrias com superfícies (mesmo fechadas), irão retornar 0.
- postgis.enable_outdb_rasters - Disponibilidade: 2.2.0 Uma opção de configuração booleana para ativar o acesso ao out-db raster bands.
- postgis.gdal_datapath - Disponibilidade: 2.2.0 Uma opção de configuração para designar o valor da opção GDAL_DATA do GDAL. Se não funcionar, a variável ambiental GDAL_DATA é usada.
- postgis.gdal_enabled_drivers - Disponibilidade: 2.2.0 Uma opção de configuração para estabelecer os drivers GDAL ativados no ambiente POstGIS. Afeta a variável GDAL_SKIP do GDAL.
- |=| - Disponibilidade: 2.2.0. Index suportado disponível somente para PostgreSQL 9.5+ Retorna a distância entre As trajetórias A e B ao ponto de aproximação mais perto.

Functions enhanced in PostGIS 2.2

- <-> - melhorias: 2.2.0 -- Verdadeiro comportamento KNN ("vizinho mais perto de K") para geometria e geografia para PostgreSQL 9.5+. Note que para geografia o KNN é baseado em esfera ao invés de esferoide. Para o PostgreSQL 9.4 ou menor, o suporte para geografia é novo, mas só suporta caixa centroide. Retorna a distância 2D entre A e B.
- ST_Area - Melhorias: 2.2.0 - medição em esferoides desempenhada com GeographicLib para uma melhor precisão e força. Requer Proj >= 4.9.0 para tirar vantagem da nova característica. Retorna o centro geométrico de uma geometria.
- ST_AsX3D - Melhorias: 2.2.0: Suporte para GeoCoordinates e eixos (x/y, long/lat) lançando. Observe as opções para mais detalhes. Retorna uma geometria em X3D nó xml formato do elemento: ISO-IEC-19776-1.2-X3DEncodings-XML
- ST_Azimuth - Melhorias: 2.2.0 medição em esferoides desempenhada com GeographicLib para uma melhor precisão e força. Requer Proj >= 4.9.0 para tirar vantagem da nova característica. Retorna a menor linha 2-dimensional entre duas geometrias
- ST_Distance - Melhorias: 2.2.0 - medição em esferoides desempenhada com GeographicLib para uma melhor precisão e força. Requer Proj >= 4.9.0 para tirar vantagem da nova característica. Retorna a linha 3-dimensional mais longa entre duas geometrias
- ST_Scale - Enhanced: 2.2.0 support for scaling all dimension (factor parameter) was introduced. Scales a geometry by given factors.
- ST_Split - Enhanced: 2.2.0 support for splitting a line by a multiline, a multipoint or (multi)polygon boundary was introduced. Returns a collection of geometries created by splitting a geometry by another geometry.
- ST_Summary - Melhorias: 2.2.0 Suporte para TIN e Curvas adicionado Retorna um texto resumo dos conteúdos da geometria.

Functions changed in PostGIS 2.2

- <-> - Alterações: 2.2.0 -- Para usuários do PostgreSQL 9.5, a sintaxe Hybrid antiga pode ser ais lenta, então, você vai querer se livrar daquele hack se você está executando seu código só no PostGIS 2.2+ 9.5+. Veja os exemplos abaixo. Retorna a distância 2D entre A e B.
- ST_3DClosestPoint - Alterações: 2.2.0 - se 2 geometrias 2D são entradas, um ponto 2D retorna (em vez do antigo comportamento assumindo 0 para Z perdido). Em caso de 2D e 3D, o Z não é mais 0 para Z perdido. Retorna o ponto 3 dimensional em g1 que é o mais próximo de g2. Este é o primeiro ponto da linha mais curta em três dimensões.

- ST_3DDistance - Alterações: 2.2.0 - Em caso de 2D e 3D, o Z não é mais 0 para Z perdido. Para tipo geometria, retorna a menor distância cartesiana 3-dimensional (baseado no sistema de referência espacial) entre duas geometrias em unidades projetadas.
- ST_3DLongestLine - Alterações: 2.2.0 - se 2 geometrias 2D são entradas, um ponto 2D retorna (em vez do antigo comportamento assumindo 0 para Z perdido). Em caso de 2D e 3D, o Z não é mais 0 para Z perdido. Retorna a linha 3-dimensional mais longa entre duas geometrias
- ST_3DMaxDistance - Alterações: 2.2.0 - Em caso de 2D e 3D, o Z não é mais 0 para Z perdido. Para tipo de geometria retorna a maior distância 3-dimensional cartesiana (baseada na referência espacial) entre duas geometrias em unidade projetadas.
- ST_3DShortestLine - Alterações: 2.2.0 - se 2 geometrias 2D são entradas, um ponto 2D retorna (em vez do antigo comportamento assumindo 0 para Z perdido). Em caso de 2D e 3D, o Z não é mais 0 para Z perdido. Retorna a menor linha 3-dimensional entre duas geometrias
- ST_DistanceSphere - Alterações: 2.2.0 Em versões anteriores era chamada de ST_Distance_Sphere Retorna a menor distância entre duas geometrias lon/lat dado um esferoide específico. As versões anteriores a 1.5 só suportam pontos.
- ST_DistanceSpheroid - Alterações: 2.2.0 Em versões anteriores era chamada de ST_Distance_Spheroid Retorna a menor distância entre duas geometrias lon/lat dado um esferoide específico. As versões anteriores a 1.5 só suportam pontos.
- ST_Equals - Changed: 2.2.0 Returns true even for invalid geometries if they are binary equal Tests if two geometries include the same set of points
- ST_LengthSpheroid - Alterações: 2.2.0 Em versões anteriores era chamada de ST_Length_Spheroid e costumava ter um heterônimo ST_3DLength_Spheroid Retorna o centro geométrico de uma geometria.
- ST_MemSize - Changed: 2.2.0 name changed to ST_MemSize to follow naming convention. Retorna o tipo de geometria de valor ST_Geometry.
- ST_PointInsideCircle - Changed: 2.2.0 In prior versions this was called ST_Point_Inside_Circle Tests if a point geometry is inside a circle defined by a center and radius

12.12.10 PostGIS Functions new or enhanced in 2.1

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 2.1

- ST_3DArea - Disponibilidade: 2.1.0 Computa a área de geometrias de superfície 3D. Irá retornar 0 para sólidos.
- ST_3DIntersection - Disponibilidade: 2.1.0 Representar intersecção 3D
- ST_Box2dFromGeoHash - Disponibilidade: 2.1.0 Retorna uma CAIXA2D de uma string GeoHash.
- ST_DelaunayTriangles - Disponibilidade: 2.1.0 Returns the Delaunay triangulation of the vertices of a geometry.
- ST_Extrude - Disponibilidade: 2.1.0 Extrude uma superfície a um volume relacionado
- ST_ForceLHR - Disponibilidade: 2.1.0 Orientação força LHR
- ST_GeomFromGeoHash - Disponibilidade: 2.1.0 Retorna uma geometria de uma string GeoHash.
- ST_MinkowskiSum - Disponibilidade: 2.1.0 Representar soma Minkowski
- ST_Orientation - Disponibilidade: 2.1.0 Determine orientação da superfície
- ST_PointFromGeoHash - Disponibilidade: 2.1.0 Retorna um ponto de uma string GeoHash.
- ST_StraightSkeleton - Disponibilidade: 2.1.0 Calcule um esqueleto em linha reta de uma geometria
- ST_Tesselate - Disponibilidade: 2.1.0 Representa superfície tesselação de um polígono ou superfície poliédrica e retorna como uma TIN ou coleção de TINS

- postgis.backend - Disponibilidade: 2.1.0 O backend para fazer a manutenção de uma função onde GEOS e SFCGAL sobrepõe. Opções: geos ou sfcgal. Padrão para geos.
- postgis_sfcgal_version - Disponibilidade: 2.1.0 retorna a versão do SFCGAL em uso

Functions enhanced in PostGIS 2.1

- ST_AsGML - Melhorias: 2.1.0 suporte para id foi introduzido, para GML 3. Retorna a geometria como uma versão GML com 2 ou 3 elementos.
- ST_Boundary - Melhorias: 2.1.0 suporte para Triângulo foi introduzido Retorna o encerramento da borda combinatória dessa geometria.
- ST_DWithin - Enhanced: 2.1.0 improved speed for geography. See Making Geography faster for details. Tests if two geometries are within a given distance
- ST_DWithin - Enhanced: 2.1.0 support for curved geometries was introduced. Tests if two geometries are within a given distance
- ST_Distance - Melhorias: 2.1.0 velocidade melhorada para geografia. Veja Making Geography faster para mais detalhes. Retorna a linha 3-dimensional mais longa entre duas geometrias
- ST_Distance - Melhorias: 2.1.0 - suporte para geometrias curvas foi introduzido. Retorna a linha 3-dimensional mais longa entre duas geometrias
- ST_MakeValid - Enhanced: 2.1.0, added support for GEOMETRYCOLLECTION and MULTIPOINT. Attempts to make an invalid geometry valid without losing vertices.
- ST_NumPoints - Enhanced: 2.1.0 Faster speed. Reimplemented as native-C. Retorna um texto resumo dos conteúdos da geometria.
- ST_Segmentize - Melhorias: 2.1.0 suporte para geografia foi introduzido. Returns a modified geometry/geography having no segment longer than a given distance.
- ST_Summary - melhorias: 2.1.0 Bandeira S para indicar se existe um sistema de referência espacial conhecido Retorna um texto resumo dos conteúdos da geometria.

Functions changed in PostGIS 2.1

- ST_EstimatedExtent - Changed: 2.1.0. Up to 2.0.x this was called ST_Estimated_Extent. Returns the estimated extent of a spatial table.
- ST_Force2D - Alterado: 2.1.0. Até versão 2.0.x isto era chamado de ST_Force_2D. Força a geometria para o modo de 2 dimensões.
- ST_Force3D - Alterado: 2.1.0. Até versão 2.0.x isto era chamado de ST_Force_3D. Força a geometria para um modo XYZ. Este é um apelido para a função ST_Force_3DZ.
- ST_Force3DM - Alterado: 2.1.0. Até a versão 2.0.x esta função era chamada de ST_Force_3DM. Força as geometrias para o modo XYM.
- ST_Force3DZ - Alterado: 2.1.0. Até versão 2.0.x isto era chamado de ST_Force_3DZ. Força as geometrias para o modo XYZ.
- ST_Force4D - Alterado: 2.1.0. Até a versão 2.0.x esta função era chamada ST_Force_4D. Força as geometrias para o modo XYZM.
- ST_ForceCollection - Alterado: 2.1.0. Até a versão 2.0.x esta função era chamada de ST_Force_Collection. Converte a geometria para um GEOMETRYCOLLECTION.
- ST_LineInterpolatePoint - Alterações: 2.1.0 para 2.0.x foi chamada ST_Line_Interpolate_Point. Returns a point interpolated along a line at a fractional location.

- ST_LineLocatePoint - Alterações: 2.1.0 para 2.0.x foi chamada ST_Line_Locate_Point. Returns the fractional location of the closest point on a line to a point.
- ST_LineSubstring - Alterações: 2.1.0 para 2.0.x foi chamada ST_Line_Substring. Returns the part of a line between two fractional locations.
- ST_Segmentize - Changed: 2.1.0 As a result of the introduction of geography support, the usage ST_Segmentize('LINESTRING(1 2, 3 4)', 0.5) causes an ambiguous function error. The input needs to be properly typed as a geometry or geography. Use ST_GeomFromText, ST_GeogFromText or a cast to the required type (e.g. ST_Segmentize('LINESTRING(1 2, 3 4)'::geometry, 0.5)) Returns a modified geometry/geography having no segment longer than a given distance.

12.12.11 PostGIS Functions new or enhanced in 2.0

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 2.0

- &&& - Disponibilidade: 2.0.0 Retorna VERDADE se a caixa limitadora n-D de A intersecta a caixa limitadora n-D de B.
- <#> - Disponibilidade: 2.0.0 -- KNN só está disponível para PostgreSQL 9.1+ Retorna a distância 2D entre as caixas limitadoras de A e B.
- <-> - Disponibilidade: 2.0.0 -- O KNN mais fraco fornece vizinho mais próximos baseados em distâncias centroides de geometrias, ao invés de distâncias reais. Resultados corretos para pontos, incorretos para todos os outros tipos. Disponível para PostgreSQL 9.1+ Retorna a distância 2D entre A e B.
- ST_3DClosestPoint - Disponibilidade: 2.0.0 Retorna o ponto 3 dimensional em g1 que é o mais próximo de g2. Este é o primeiro ponto da linha mais curta em três dimensões.
- ST_3DDFullyWithin - Availability: 2.0.0 Tests if two 3D geometries are entirely within a given 3D distance
- ST_3DDWithin - Availability: 2.0.0 Tests if two 3D geometries are within a given 3D distance
- ST_3DDistance - Disponibilidade: 2.0.0 Para tipo geometria, retorna a menor distância cartesiana 3-dimensional (baseado no sistema de referência espacial) entre duas geometrias em unidades projetadas.
- ST_3DIntersects - Availability: 2.0.0 Tests if two geometries spatially intersect in 3D - only for points, linestrings, polygons, polyhedral surface (area)
- ST_3DLongestLine - Disponibilidade: 2.0.0 Retorna a linha 3-dimensional mais longa entre duas geometrias
- ST_3DMaxDistance - Disponibilidade: 2.0.0 Para tipo de geometria retorna a maior distância 3-dimensional cartesiana (baseada na referência espacial) entre duas geometrias em unidade projetadas.
- ST_3DShortestLine - Disponibilidade: 2.0.0 Retorna a menor linha 3-dimensional entre duas geometrias
- ST_AsLatLonText - Disponibilidade: 2.0 Retorna a representação de Graus, Minutos, Segundos do ponto dado.
- ST_AsX3D - Disponibilidade: 2.0.0: ISO-IEC-19776-1.2-X3DEncodings-XML Retorna uma geometria em X3D nó xml formato do elemento: ISO-IEC-19776-1.2-X3DEncodings-XML
- ST_CollectionHomogenize - Disponibilidade: 2.0.0 Returns the simplest representation of a geometry collection.
- ST_ConcaveHull - Disponibilidade: 2.0.0 Computes a possibly concave geometry that contains all input geometry vertices
- ST_FlipCoordinates - Disponibilidade: 2.0.0 Returns a version of a geometry with X and Y axis flipped.
- ST_GeomFromGeoJSON - Disponibilidade: 2.0.0 requer - JSON-C >= 0.9 Utiliza como entrada uma representação geojson de uma geometria e como saída um objeto de geometria PostGIS
- ST_InterpolatePoint - Disponibilidade: 2.0.0 Retorna o valor da dimensão de medida da geometria no ponto fechado para o ponto fornecido.

- ST_IsValidDetail - Availability: 2.0.0 Returns a valid_detail row stating if a geometry is valid or if not a reason and a location.
- ST_IsValidReason - Availability: 2.0 version taking flags. Returns text stating if a geometry is valid, or a reason for invalidity.
- ST_MakeLine - Disponibilidad: 2.0.0 - Suporte para elementos de entrada linestring foi introduzido Cria uma Linestring de ponto, multiponto ou linha das geometrias.
- ST_MakeValid - Availability: 2.0.0 Attempts to make an invalid geometry valid without losing vertices.
- ST_Node - Availability: 2.0.0 Nodes a collection of lines.
- ST_NumPatches - Disponibilidade: 2.0.0 Retorna o número de faces em uma superfícies poliédrica. Retornará nulo para geometrias não poliédricas.
- ST_OffsetCurve - Disponibilidade: 2.0 Returns an offset line at a given distance and side from an input line.
- ST_PatchN - Disponibilidade: 2.0.0 Retorna o tipo de geometria de valor ST_Geometry.
- ST_Perimeter - Disponibilidade 2.0.0: Suporte para geografia foi introduzido Returns the length of the boundary of a polygonal geometry or geography.
- ST_Project - Disponibilidade: 2.0.0 Returns a point projected from a start point by a distance and bearing (azimuth).
- ST_RelateMatch - Availability: 2.0.0 Tests if a DE-9IM Intersection Matrix matches an Intersection Matrix pattern
- ST_SharedPaths - Disponibilidade: 2.0.0 Retorna uma coleção contendo caminhos compartilhados pelas duas linestrings/multilinestrings de entrada.
- ST_Snap - Disponibilidade: 2.0.0 Rompe segmentos e vértices de geometria de entrada para vértices de uma geometria de referência.
- ST_Split - Availability: 2.0.0 requires GEOS Returns a collection of geometries created by splitting a geometry by another geometry.
- ST_UnaryUnion - Availability: 2.0.0 Computes the union of the components of a single geometry.

Functions enhanced in PostGIS 2.0

- && - Melhorias: 2.0.0 suporte a superfícies poliédricas foi introduzido. Retorna VERDADE se a caixa limitadora 2D de A intersecta a caixa limitadora 2D de B.
- AddGeometryColumn - Melhorias: 2.0.0 argumento use_typmod introduzido. Padrões para criar colunas de geometria typmod ao invés das baseadas em obstáculos. Remove uma coluna geometria de uma spatial table.
- Box2D - Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced. Returns a BOX2D representing the 2D extent of a geometry.
- Box3D - Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced. Returns a BOX3D representing the 3D extent of a geometry.
- Populate_Geometry_Columns - Melhorias: 2.0.0 use_typmod argumento opcional foi introduzido, permitindo controlar se as colunas forem criadas com modificadores de tipo ou com verificação de restrições. Ensures geometry columns are defined with type modifiers or have appropriate spatial constraints.
- ST_3DExtent - Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced. Aggregate function that returns the 3D bounding box of geometries.
- ST_Affine - Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced. Apply a 3D affine transformation to a geometry.
- ST_Area - Melhorias: 2.0.0 - suporte a superfícies 2D poliédricas foi introduzido. Retorna o centro geométrico de uma geometria.
- ST_AsBinary - Melhorias: 2.0.0 suporte para superfícies poliédricas, triângulos e TINs introduzido. Return the OGC/ISO Well-Known Binary (WKB) representation of the geometry/geography without SRID meta data.

- ST_AsBinary - Melhorias: 2.0.0 suporte para maiores dimensões de coordenadas foi introduzido. Return the OGC/ISO Well-Known Binary (WKB) representation of the geometry/geography without SRID meta data.
- ST_AsBinary - Melhorias: 2.0.0 suporte para edian especificando com geografia foi introduzido. Return the OGC/ISO Well-Known Binary (WKB) representation of the geometry/geography without SRID meta data.
- ST_AsEWKB - Melhorias: 2.0.0 suporte para superfícies poliédricas, triângulos e TINs introduzido. Return the Extended Well-Known Binary (EWKB) representation of the geometry with SRID meta data.
- ST_AsEWKT - Melhorias: 2.0.0 suporte para geografia, superfícies poliédricas, triângulos e TIN foi introduzido. Retorna a representação de texto bem conhecida (WKT) da geometria com os meta dados SRID.
- ST_AsGML - Melhorias: 2.0.0 prefixo suportado foi introduzido. A opção 4 para o GML3 foi introduzida para permitir a utilização da LineString em vez da tag Curva para linhas. O suporte GML3 para superfícies poliédricas e TINS foi introduzidos. A Opção 32 foi introduzida para gerar a caixa. Retorna a geometria como uma versão GML com 2 ou 3 elementos.
- ST_AsKML - Melhorias: 2.0.0 - Adiciona namespace prefixo. O padrão é não ter nenhum prefixo Retorna a geometria como uma versão GML com 2 ou 3 elementos.
- ST_Azimuth - Melhorias: 2.0.0 suporte para geografia foi introduzido. Retorna a menor linha 2-dimensional entre duas geometrias
- ST_Dimension - Melhorias: 2.0.0 suporte para superfícies poliédricas e TINs foi introduzido. Não abre mais exceção se uma geometria vazia é dada. Retorna a dimensão da coordenada do valor ST_Geometry.
- ST_Dump - Melhorias: 2.0.0 suporte para superfícies poliédricas, triângulos e TIN introduzido. Returns a set of geometry_dump rows for the components of a geometry.
- ST_Expand - Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced. Returns a bounding box expanded from another bounding box or a geometry.
- ST_Extent - Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced. Aggregate function that returns the bounding box of geometries.
- ST_Force2D - Melhorias: 2.0.0 suporte a superfícies polihédricas foi introduzido. Força a geometria para o modo de 2 dimensões.
- ST_Force3D - Melhorias: 2.0.0 suporte a superfícies polihédricas foi introduzido. Força a geometria para um modo XYZ. Este é um apelido para a função ST_Force_3DZ.
- ST_Force3DZ - Melhorias: 2.0.0 suporte a superfícies polihédricas foi introduzido. Força as geometrias para o modo XYZ.
- ST_ForceCollection - Melhorias: 2.0.0 suporte a superfícies polihédricas foi introduzido. Converte a geometria para um GEOMETRYCOLLECTION.
- ST_ForceRHR - Melhorias: 2.0.0 suporte a superfícies polihédricas foi introduzido. Força a orientação dos vértices em um polígono a seguir a regra da mão direita.
- ST_GMLToSQL - Melhorias: 2.0.0 suporte para superfícies poliédricas e TIN foi introduzido. Retorna um valor ST_Geometry específico da representação GML. Esse é um heterônimo para ST_GeomFromGML
- ST_GMLToSQL - Melhorias: 2.0.0 parâmetro opcional padrão srid adicionado. Retorna um valor ST_Geometry específico da representação GML. Esse é um heterônimo para ST_GeomFromGML
- ST_GeomFromEWKB - Melhorias: 2.0.0 suporte para superfícies poliédricas e TIN foi introduzido. Retorna um valor ST_Geometry especifico da representação binária estendida bem conhecida (EWKB).
- ST_GeomFromEWKT - Melhorias: 2.0.0 suporte para superfícies poliédricas e TIN foi introduzido. Retorna um valor ST_Geometry especifico da representação de texto estendida bem conhecida (EWKT).
- ST_GeomFromGML - Melhorias: 2.0.0 suporte para superfícies poliédricas e TIN foi introduzido. Utiliza como entrada uma representação GML de geometria e como saída um objeto de geometria PostGIS
- ST_GeomFromGML - Melhorias: 2.0.0 parâmetro opcional padrão srid adicionado. Utiliza como entrada uma representação GML de geometria e como saída um objeto de geometria PostGIS

- ST_GeometryN - Melhorias: 2.0.0 suporte para superfícies poliédricas, triângulos e TIN introduzido. Retorna o tipo de geometria de valor ST_Geometry.
- ST_GeometryType - Melhorias: 2.0.0 suporte a superfícies poliédricas foi introduzido. Retorna o tipo de geometria de valor ST_Geometry.
- ST_IsClosed - Melhorias: 2.0.0 suporte a superfícies poliédricas foi introduzido. Retorna VERDADEIRO se os pontos de começo e fim da LINESTRING são coincidentes. Para superfície poliédrica está fechada (volumétrica).
- ST_MakeEnvelope - Melhorias: 2.0: Habilidade para especificar um pacote sem especificar um SRID foi introduzida. Cria um polígono retangular formado a partir dos mínimos e máximos dados. Os valores de entrada devem ser em SRS especificados pelo SRID.
- ST_MakeValid - Enhanced: 2.0.1, speed improvements Attempts to make an invalid geometry valid without losing vertices.
- ST_NPoints - Melhorias: 2.0.0 suporte a superfícies poliédricas foi introduzido. Retorna o número de pontos (vértices) em uma geometria.
- ST_NumGeometries - Melhorias: 2.0.0 suporte para superfícies poliédricas, triângulos e TIN introduzido. Retorna o número de pontos em uma geometria. Funciona para todas as geometrias.
- ST_NumPoints - Melhorias: 2.0.0 suporte para superfícies poliédricas, triângulos e TIN introduzido. Retorna um texto resumo dos conteúdos da geometria.
- ST_Relate - Enhanced: 2.0.0 - added support for specifying boundary node rule. Tests if two geometries have a topological relationship matching an Intersection Matrix pattern, or computes their Intersection Matrix
- ST_Rotate - Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced. Rotates a geometry about an origin point.
- ST_Rotate - Enhanced: 2.0.0 additional parameters for specifying the origin of rotation were added. Rotates a geometry about an origin point.
- ST_RotateX - Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced. Rotates a geometry about the X axis.
- ST_RotateY - Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced. Rotates a geometry about the Y axis.
- ST_RotateZ - Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced. Rotates a geometry about the Z axis.
- ST_Scale - Enhanced: 2.0.0 support for Polyhedral surfaces, Triangles and TIN was introduced. Scales a geometry by given factors.
- ST_ShiftLongitude - Melhorias: 2.0.0 suporte para superfícies poliédricas e TIN foi introduzido. Shifts the longitude coordinates of a geometry between -180..180 and 0..360.
- ST_Summary - Melhorias: 2.0.0 suporte para geografia adicionado Retorna um texto resumo dos conteúdos da geometria.
- ST_Transform - Enhanced: 2.0.0 support for Polyhedral surfaces was introduced. Return a new geometry with coordinates transformed to a different spatial reference system.
- Tipo de geometria - Melhorias: 2.0.0 suporte para superfícies poliédricas, triângulos e TIN introduzido. Retorna o tipo de geometria de valor ST_Geometry.

Functions changed in PostGIS 2.0

- AddGeometryColumn - Alterado: 2.0.0 Essa função não atualiza mais a geometry_columns desde que ela é a view que lê dos catálogos de sistema. Por padrão, isso não cria restrições, mas usa a construção no comportamento do tipo modificador do PostgreSQL. Então, por exemplo, construir uma coluna wgs84 POINT com essa função é equivalente a: ALTER TABLE some_table ADD COLUMN geom geometry(Point,4326); Remove uma coluna geometria de uma spatial table.

- AddGeometryColumn - Alterado: 2.0.0 Se você exige o comportamento antigo de restrições use o padrão use_typmod, mas configure isso para falso. Remove uma coluna geometria de uma spatial table.
- AddGeometryColumn - Alterações: 2.0.0 Views não podem ser registradas manualmente mais em geometry_columns, porém as views construídas contra as geometrias typmod tables e usadas sem as funções wrapper irão se registrar corretamente, porque elas herdam um comportamento typmod da table column mãe. As views que usam funções geométricas que fazem outras geometrias saírem, precisarão de ser lançadas para as geometrias typmod, para essas colunas serem registradas corretamente em geometry_columns. Use . Remove uma coluna geometria de uma spatial table.
- DropGeometryColumn - Alterações: 2.0.0 Essa função é fornecida para compatibilidade atrasada. Desde que geometry_columns é uma view contra os sistemas catalogados, você pode derrubar uma coluna geométrica como qualquer outra table column usando ALTERAR TABLE Remove uma coluna geometria de uma spatial table.
- DropGeometryTable - Alterações: 2.0.0 Essa função é fornecida para compatibilidade atrasada. Desde que geometry_columns é uma view contra os sistemas catalogados, você pode derrubar uma table com colunas geométricas como qualquer outra table usando DERRUBAR TABLE Derruba uma table e todas suas referências em geometry_columns.
- Populate_Geometry_Columns - Alterações: 2.0.0 Por padrão, utilize modificadores de tipo ao invés de verificar restrições para restringir os tipos de geometria. Você pode verificar restrições de comportamento ao invés de usar o novo use_typmod e configurá-lo para falso. Ensures geometry columns are defined with type modifiers or have appropriate spatial constraints.
- ST_3DExtent - Changed: 2.0.0 In prior versions this used to be called ST_Extent3D Aggregate function that returns the 3D bounding box of geometries.
- ST_3DLength - Alterações: 2.0.0 Nas versões anteriores era chamado de ST_Length3D Retorna o centro geométrico de uma geometria.
- ST_3DMakeBox - Changed: 2.0.0 In prior versions this used to be called ST_MakeBox3D Creates a BOX3D defined by two 3D point geometries.
- ST_3DPerímetro - Alterações: 2.0.0 Nas versões anteriores era chamado de ST_Perimeter3D Retorna o centro geométrico de uma geometria.
- ST_AsBinary - Alterações: 2.0.0 Entrada para esta função não pode ser desconhecida -- deve ser geometria. Construções como ST_AsBinary('POINT(1 2)') não são mais válidas e você terá n st_asbinary(desconhecido) não é um erro único. Códigos assim, precisam ser alterados para ST_AsBinary('POINT(1 2)'::geometry);. Se não for possível, instale: legacy.sql. Return the OGC/ISO Well-Known Binary (WKB) representation of the geometry/geography without SRID meta data.
- ST_AsGML - Alterações: 2.0.0 use argumentos nomeados por padrão Retorna a geometria como uma versão GML com 2 ou 3 elementos.
- ST_AsGeoJSON - Alterações: 2.0.0 suporte padrão args e args nomeados. Return a geometry as a GeoJSON element.
- ST_AsSVG - Alterações: 2.0.0 para usar args padrão e suporta args nomeados Returns SVG path data for a geometry.
- ST_EndPoint - Alterações: 2.0.0 não funciona mais com geometrias de multilinestrings. Em verões mais antigas do PostGIS -- uma linha multilinestring sozinha trabalharia normalmente com essa função e voltaria o ponto de início. Na 2.0.0 ela retorna NULA como qualquer outra multilinestring. O antigo comportamento não foi uma característica documentada, mas as pessoas que consideravam que tinham seus dados armazenados como uma LINESTRING, agora podem experimentar essas que retornam NULAS em 2.0. Retorna o número de pontos em um valor ST_LineString ou ST_CircularString.
- ST_GeomFromText - Alterações: 2.0.0 Nas primeiras versões do PostGIS, ST_GeomFromText('GEOMETRYCOLLECTION(EMPT foi permitida. Ela agora é ilegal no PostGIS 2.0.0 para melhor se adequar aos padrões SQL/MM. Ela deverá se escrita como ST_GeomFromText('GEOMETRYCOLLECTION EMPTY') Retorna um valor ST_Geometry específico da representação de texto bem conhecida (WKT).
- ST_GeometryN - Alterações: 2.0.0. Versões anteriores voltariam NULAS para geometrias únicas. Isso foi alterado para volrtar a geometria para o caso ST_GeometryN(..,1). Retorna o tipo de geometria de valor ST_Geometry.
- ST_IsEmpty - Alterações: 2.0.0 Nas versões anteriores do PostGIS ST_GeomFromText('GEOMETRYCOLLECTION(EMPTY)') era permitido. Agora isso é ilegal no PostGIS 2.0.0 para se adequar aos padrões SQL/MM. Tests if a geometry is empty.

- ST_Length - Alterações: 2.0.0 Quebrando a mudança -- nas versões anteriores aplicar isto a um MULTI/POLÍGONO de tipo de geografia lhe daria o perímetro do POLÍGONO/MULTIPOLÍGONO. Na 2.0.0 isso é alterado para retornar 0 a estar na linha com o comportamento da geometria. Por favor, utilize a ST_Perimeter se quiser o perímetro de um polígono Retorna o centro geométrico de uma geometria.
- ST_LocateAlong - Alterações: 2.0.0 nas versões anteriores era chamado de ST_Locate_Along_Measure. O nome antigo foi menosprezado e será removido no futuro, mas ainda está disponível. Returns the point(s) on a geometry that match a measure value.
- ST_LocateBetween - Alterações: 2.0.0 nas versões anteriores era chamado de ST_Locate_Along_Measure. O nome antigo foi menosprezado e será removido no futuro, mas ainda está disponível. Returns the portions of a geometry that match a measure range.
- ST_NumGeometries - Alterações: 2.0.0 Em versões anteriores retornaria NULO se a geometria não fosse do tipo coleção/-MULTI. 2.0.0+ agora retorna 1 para geometrias únicas ex: POLÍGONO, LINESTRING, PONTO. Retorna o número de pontos em uma geometria. Funciona para todas as geometrias.
- ST_NumInteriorRings - Alterações: 2.0.0 - nas versões anteriores isso permitiria um MULTIPOLÍGONO, retornando o número de anéis interiores do primeiro POLÍGONO. Retorna o número de anéis interiores de um polígono.
- ST_PointN - Alterações: 2.0.0 não funciona mais com geometrias multilinestrings únicas. Em verões mais antigas do PostGIS -- uma única linha multilinestring trabalharia normalmente e retornaria o ponto inicial. Na 2.0.0 só retorna NULA como qualquer outra multilinestring. Retorna o número de pontos em um valor ST_LineString ou ST_CircularString.
- ST_StartPoint - Alterações: 2.0.0 não funciona mais com geometrias de multilinestrings. Em verões mais antigas do PostGIS -- uma linha multilinestring sozinha trabalharia normalmente com essa função e voltaria o ponto de início. Na 2.0.0 ela retorna NULA como qualquer outra multilinestring. O antigo comportamento não foi uma característica documentada, mas as pessoas que consideravam que tinham seus dados armazenados como uma LINESTRING, agora podem experimentar essas que retornam NULAS em 2.0. Returns the first point of a LineString.

12.12.12 PostGIS Functions new or enhanced in 1.5

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 1.5

- && - Disponibilidade: 1.5.0 Suporte para geografia foi introduzido Retorna VERDADE se a caixa limitadora 2D de A intersecta a caixa limitadora 2D de B.
- PostGIS_LibXML_Version - Availability: 1.5 Returns the version number of the libxml2 library.
- ST_AddMeasure - Disponibilidade: 1.5.0 Interpolates measures along a linear geometry.
- ST_AsBinary - Disponibilidade: 1.5.0 suporte para geografia foi introduzido. Return the OGC/ISO Well-Known Binary (WKB) representation of the geometry/geography without SRID meta data.
- ST_AsGML - Disponibilidade: 1.5.0 suporte para geografia foi introduzido. Retorna a geometria como uma versão GML com 2 ou 3 elementos.
- ST_AsGeoJSON - Disponibilidade: 1.5.0 suporte para geografia foi introduzido. Return a geometry as a GeoJSON element.
- ST_AsText - Disponibilidade: 1.5 - suporte para geografia foi introduzido. Retorna a representação de texto bem conhecida (WKT) da geometria/geografia sem os meta dados do SRID.
- ST_Buffer - Availability: 1.5 - ST_Buffer was enhanced to support different endcaps and join types. These are useful for example to convert road linestrings into polygon roads with flat or square edges instead of rounded edges. Thin wrapper for geography was added. Computes a geometry covering all points within a given distance from a geometry.
- ST_ClosestPoint - Disponibilidade: 1.5.0 Returns the 2D point on g1 that is closest to g2. This is the first point of the shortest line from one geometry to the other.

- ST_CollectionExtract - Disponibilidade: 1.5.0 Given a geometry collection, returns a multi-geometry containing only elements of a specified type.
- ST_Covers - Availability: 1.5 - support for geography was introduced. Tests if every point of B lies in A
- ST_DFullyWithin - Availability: 1.5.0 Tests if two geometries are entirely within a given distance
- ST_DWithin - Availability: 1.5.0 support for geography was introduced Tests if two geometries are within a given distance
- ST_Distance - Disponibilidade: 1.5.0 suporte de geografia foi introduzido em 1.5. Melhorias na velocidade para planar para lidar melhor com mais ou maiores vértices de geometrias. Retorna a linha 3-dimensional mais longa entre duas geometrias
- ST_DistanceSphere - Disponibilidade: 1.5 - suporte para outros tipos de geometria além de pontos foi introduzido. As versões anteriores só funcionam com pontos. Retorna a menor distância entre duas geometrias lon/lat dado um esferoide específico. As versões anteriores a 1.5 só suportam pontos.
- ST_DistanceSpheroid - Disponibilidade: 1.5 - suporte para outros tipos de geometria além de pontos foi introduzido. As versões anteriores só funcionam com pontos. Retorna a menor distância entre duas geometrias lon/lat dado um esferoide específico. As versões anteriores a 1.5 só suportam pontos.
- ST_Envelope - Disponibilidade: 1.5.0 comportamento alterado para saída de precisão dupla ao invés de float4 Retorna uma geometria representando a precisão da dobrada (float8) da caixa limitada da geometria fornecida.
- ST_Expand - Availability: 1.5.0 behavior changed to output double precision instead of float4 coordinates. Returns a bounding box expanded from another bounding box or a geometry.
- ST_GMLToSQL - Disponibilidade:1.5, requer libxml2 1.6+ Retorna um valor ST_Geometry específico da representação GML. Esse é um heterônimo para ST_GeomFromGML
- ST_GeomFromGML - Disponibilidade:1.5, requer libxml2 1.6+ Utiliza como entrada uma representação GML de geometria e como saída um objeto de geometria PostGIS
- ST_GeomFromKML - Availability: 1.5, requires libxml2 2.6+ Utiliza como entrada uma representação KML de geometria e como saída um objeto de geometria PostGIS
- ST_HausdorffDistance - Disponibilidade: 1.5.0 Retorna a menor linha 3-dimensional entre duas geometrias
- ST_Intersection - Availability: 1.5 support for geography data type was introduced. Computes a geometry representing the shared portion of geometries A and B.
- ST_Intersects - Availability: 1.5 support for geography was introduced. Tests if two geometries intersect (they have at least one point in common)
- ST_Length - Disponibilidade: 1.5.0 suporte para geografia foi introduzido em 1.5. Retorna o centro geométrico de uma geometria.
- ST_LongestLine - Disponibilidade: 1.5.0 Retorna a linha 3-dimensional mais longa entre duas geometrias
- ST_MakeEnvelope - Disponibilidade: 1.5 Cria um polígono retangular formado a partir dos mínimos e máximos dados. Os valores de entrada devem ser em SRS especificados pelo SRID.
- ST_MaxDistance - Disponibilidade: 1.5.0 Retorna a maior distância 2-dimensional entre duas geometrias em unidades projetadas.
- ST_NumPoints - Disponibilidade: 1.2.2 Retorna um texto resumo dos conteúdos da geometria.
- ST_ShortestLine - Disponibilidade: 1.5.0 Retorna a menor linha 2-dimensional entre duas geometrias
- ~= - Disponibilidade: 1.5.0 comportamento alterado Retorna VERDADE se a caixa limitadora de A é a mesma de B.

12.12.13 PostGIS Functions new or enhanced in 1.4

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 1.4

- Populate_Geometry_Columns - Disponibilidade: 1.4.0 Ensures geometry columns are defined with type modifiers or have appropriate spatial constraints.
- ST_ContainsProperly - Availability: 1.4.0 Tests if every point of B lies in the interior of A
- ST_GeoHash - Disponibilidade: 1.4.0 Retorna uma representação GeoHash da geometria.
- ST_GeomCollFromText - Disponibilidade: 1.4.0 - ST_MakeLine(geomarray) foi introduzida. A ST_MakeLine agrega funções que foram melhoradas para lidar com mais pontos mais rápido. Creates a GeometryCollection or Multi* geometry from a set of geometries.
- ST_IsValidReason - Availability: 1.4 Returns text stating if a geometry is valid, or a reason for invalidity.
- ST_LineCrossingDirection - Availability: 1.4 Returns a number indicating the crossing behavior of two LineStrings
- ST_LocateBetweenElevations - Disponibilidade: 1.4.0 Returns the portions of a geometry that lie in an elevation (Z) range.
- ST_MakeLine - Disponibilidade: 1.4.0 - ST_MakeLine(geomarray) foi introduzida. A ST_MakeLine agrega funções que foram melhoradas para lidar com mais pontos mais rápido. Cria uma Linestring de ponto, multiponto ou linha das geometrias.
- ST_MinimumBoundingCircle - Disponibilidade: 1.4.0 Returns the smallest circle polygon that contains a geometry.
- ST_Union - Availability: 1.4.0 - ST_Union was enhanced. ST_Union(geomarray) was introduced and also faster aggregate collection in PostgreSQL. Computes a geometry representing the point-set union of the input geometries.

12.12.14 PostGIS Functions new or enhanced in 1.3

The functions given below are PostGIS functions that were added or enhanced.

Functions new in PostGIS 1.3

- ST_AsGML - Disponibilidade: 1.3.2 Retorna a geometria como uma versão GML com 2 ou 3 elementos.
- ST_AsGeoJSON - Disponibilidade: 1.3.4 Return a geometry as a GeoJSON element.
- ST_CurveToLine - Availability: 1.3.0 Converts a geometry containing curves to a linear geometry.
- ST_LineToCurve - Availability: 1.3.0 Converts a linear geometry to a curved geometry.
- ST_SimplifyPreserveTopology - Disponibilidade: 1.3.3 Returns a simplified and valid version of a geometry, using the Douglas-Peucker algorithm.

Chapter 13

Reporting Problems

13.1 Reporting Software Bugs

Reporting bugs effectively is a fundamental way to help PostGIS development. The most effective bug report is that enabling PostGIS developers to reproduce it, so it would ideally contain a script triggering it and every information regarding the environment in which it was detected. Good enough info can be extracted running `SELECT postgis_full_version()` [for PostGIS] and `SELECT version()` [for postgresql].

If you aren't using the latest release, it's worth taking a look at its release changelog first, to find out if your bug has already been fixed.

Using the PostGIS bug tracker will ensure your reports are not discarded, and will keep you informed on its handling process. Before reporting a new bug please query the database to see if it is a known one, and if it is please add any new information you have about it.

You might want to read Simon Tatham's paper about How to Report Bugs Effectively before filing a new report.

13.2 Reporting Documentation Issues

The documentation should accurately reflect the features and behavior of the software. If it doesn't, it could be because of a software bug or because the documentation is in error or deficient.

Documentation issues can also be reported to the PostGIS bug tracker.

If your revision is trivial, just describe it in a new bug tracker issue, being specific about its location in the documentation.

If your changes are more extensive, a patch is definitely preferred. This is a four step process on Unix (assuming you already have git installed):

1. Clone the PostGIS' git repository. On Unix, type:

   git clone https://git.osgeo.org/gitea/postgis/postgis.git

   This will be stored in the directory postgis

2. Make your changes to the documentation with your favorite text editor. On Unix, type (for example):

   vim doc/postgis.xml

   Note that the documentation is written in DocBook XML rather than HTML, so if you are not familiar with it please follow the example of the rest of the documentation.

3. Make a patch file containing the differences from the master copy of the documentation. On Unix, type:

   git diff doc/postgis.xml > doc.patch

4. Attach the patch to a new issue in bug tracker.

Appendix A

Apêndice

A.1 PostGIS 3.4.0beta1

2023/07/14

This version requires PostgreSQL 12 or higher, GEOS 3.6 or higher, and Proj 6.1+. To take advantage of all features, GEOS 3.12+ is needed. To take advantage of all SFCGAL featurs, SFCGAL 1.4.1+ is needed.

NOTE: GEOS 3.12.0 details at GEOS 3.12.0 release notes

Many thanks to our translation teams, in particular:

Teramoto Ikuhiro (Japanese Team)

Vincent Bre (French Team)

A.1.1 New features

5055, complete manual internationalization (Sandro Santilli)

5052, target version support in postgis_extensions_upgrade (Sandro Santilli)

5306, expose version of GEOS at compile time (Sandro Santilli)

New install-extension-upgrades command in postgis script (Sandro Santilli)

5257, 5261, 5277, Support changes for PostgreSQL 16 (Regina Obe)

5006, 705, ST_Transform: Support PROJ pipelines (Robert Coup, Koordinates)

5283, [postgis_topology] RenameTopology (Sandro Santilli)

5286, [postgis_topology] RenameTopoGeometryColumn (Sandro Santilli)

703, [postgis_raster] Add min/max resampling as options (Christian Schroeder)

5336, [postgis_topology] topogeometry cast to topoelement support (Regina Obe)

Allow singleton geometry to be inserted into Geometry(Multi*) columns (Paul Ramsey)

721, New window-based ST_ClusterWithinWin and ST_ClusterIntersectingWin (Paul Ramsey)

5397, [address_standardizer] debug_standardize_address function (Regina Obe)

5373ST_LargestEmptyCircle, exposes extra semantics on circle finding. Geos 3.9+ required(Martin Davis)

5267, ST_Project signature for geometry, and two-point signature (Paul Ramsey)

5267, ST_LineExtend for extending linestrings (Paul Ramsey)

A.1.2 Melhorias

5194, do not update system catalogs from postgis_extensions_upgrade (Sandro Santilli)

5092, reduce number of upgrade paths installed on system (Sandro Santilli)

635, honour --bindir (and --prefix) configure switch for executables (Sandro Santilli)

Honour --mandir (and --prefix) configure switch for man pages install path (Sandro Santilli)

Honour --htmldir (and --docdir and --prefix) configure switch for html pages install path (Sandro Santilli)

[postgis_topology] Speed up check of topology faces without edges (Sandro Santilli)

[postgis_topology] Speed up coincident nodes check in topology validation (Sandro Santilli)

718, ST_QuantizeCoordinates(): speed-up implementation (Even Rouault)

Repair spatial planner stats to use computed selectivity for contains/within queries (Paul Ramsey)

734, Additional metadata on Proj installation in postgis_proj_version (Paul Ramsey)

5177, allow building tools without PostgreSQL server headers (Sandro Santilli)

ST_Project signature for geometry, and two-point signature (Paul Ramsey)

4913, ST_AsSVG support for curve types CircularString, CompoundCurve, MultiCurve, and MultiSurface (Regina Obe)

5266, ST_ClosestPoint, ST_ShortestLine support for geography type (MobilityDB Esteban Zimanyi, Paul Ramsey)

A.1.3 Breaking Changes

5229, Drop support for Proj < 6.1 and PG 11 (Regina Obe)

5306, 734, postgis_full_version() and postgis_proj_version() now output more information about proj network configuration and data paths. GEOS compile-time version also shown if different from run-time (Paul Ramsey, Sandro Santilli)