PostGIS 1.5 Manual

SVN Revision (11757)

Contents

Abstract

PostGIS is an extension to the PostgreSQL object-relational database system which allows GIS (Geographic Information Systems) objects to be stored in the database. PostGIS includes support for GiST-based R-Tree spatial indexes, and functions for analysis and processing of GIS objects.

This is the manual for version 1.5.9SVN

SVN Revision (*11757*)

Chapter 1

Introduction

PostGIS is developed 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. We plan on supporting and developing PostGIS to support a range of important GIS functionality, including full OpenGIS support, advanced topological constructs (coverages, surfaces, networks), desktop user interface tools for viewing and editing GIS data, and web-based access tools.

1.1 Project Steering Committee

The PostGIS Project Steering Committee (PSC) coordinates the general direction, release cycles, documentation, and outreach efforts for the PostGIS project. In addition the PSC provides general user support, accepts and approves patches from the general PostGIS community and votes on miscellaneous issues involving PostGIS such as developer commit access, new PSC members or significant API changes.

Mark Cave-Ayland Coordinates bug fixing and maintenance effort, alignment of PostGIS with PostgreSQL releases, spatial index selectivity and binding, windows production builds, integration of new GEOS functionality, and new function enhancements.

Paul Ramsey Co-founder of PostGIS project. General bug fixing, geography support, GEOS functionality integration and alignment with GEOS releases.

Kevin Neufeld Documentation, Hudson automated build, advanced user support on PostGIS newsgroup, and postgis maintenance function enhancements.

Regina Obe Documentation, general user support on PostGIS newsgroup, windows production and experimental builds, and smoke testing new functionality or major code changes.

1.2 Contributors Past and Present

Sandro Santilli Bug fixes and maintenance and integration of new GEOS functionality. WKT Raster support.

Dave Blasby The original developer/Co-founder of PostGIS. Dave wrote the server side objects, index bindings, and many of the server side analytical functions.

Jeff Lounsbury Original development of the Shape file loader/dumper. Current PostGIS Project Owner representative.

Mark Leslie Ongoing maintenance and development of core functions. Enhanced curve support.

Olivier Courtin Input output XML (KML,GML)/GeoJSON functions and bug fixes.

Pierre Racine WKT Raster overall architecture and programming support

Mateusz Loskot WKT Raster support

Chris Hodgson General development

Nicklas Avén Distance function enhancements and additions, Windows testing, and general user support

Jorge Arevalo WKT Raster development

Stephen Frost Tiger geocoder development

Other contributors In alphabetical order: Alex Bodnaru, Alex Mayrhofer, Barbara Phillipot, Ben Jubb, Bernhard Reiter, Bruce Rindahl, Bruno Wolff III, Carl Anderson, Charlie Savage, Dane Springmeyer, David Skea, David Techer, Eduin Carrillo, IIDA Tetsushi, George Silva, Geographic Data BC, Gerald Fenoy, Gino Lucrezi, Greg Stark, Guillaume Lelarge, Klaus Foerster, Kris Jurka, Mark Sondheim, Markus Schaber, Maxime Guillaud, Maxime van Noppen, Michael Fuhr, Nikita Shulga, Norman Vine, Ralph Mason, Steffen Macke, Vincent Picavet

Important Support Libraries The GEOS geometry operations library, and the algorithmic work of Martin Davis in making it all work, ongoing maintenance and support of Mateusz Loskot, Paul Ramsey and others.

The Proj4 cartographic projection library, and the work of Gerald Evenden and Frank Warmerdam in creating and maintaining it.

1.3 More Information

- The latest software, documentation and news items are available at the PostGIS web site, http://postgis.refractions.net.
- More information about the GEOS geometry operations library is available athttp://trac.osgeo.org/geos/.
- More information about the Proj4 reprojection library is available at http://trac.osgeo.org/proj/.
- More information about the PostgreSQL database server is available at the PostgreSQL main site http://www.postgresql.org.
- More information about GiST indexing is available at the PostgreSQL GiST development site, http://www.sai.msu.su/~megera/postgres/gist/.
- More information about MapServer internet map server is available at http://mapserver.gis.umn.edu.
- The "Simple Features for Specification for SQL" is available at the OpenGIS Consortium web site: http://www.opengeospatial.org/.

Chapter 2

Installation

This chapter details the steps required to install PostGIS.

2.1 Short Version

```
tar xvfz postgis-1.5.9SVN.tar.gz
cd postgis-1.5.9SVN
./configure
make
make install
#BEGIN OPTIONAL -- this is already part of the tar
# only really need this if installing from SVN
cd doc/
make comments-install
#END OPTIONAL
createdb yourdatabase
createlang plpgsql yourdatabase
psql -d yourdatabase -f postgis.sql
psql -d yourdatabase -f postgis_comments.sql
psql -d yourdatabase -f spatial_ref_sys.sql
```

Note

NOTE: The postgis.sql and spatial_ref_sys.sql will be installed in the /share/contrib/postgis-1.5 of your PostGIS install. If you didn't install the OPTIONAL comments section, you will need to manually copy the postgis_comments.sql file from the doc folder of your source install to your /share/contrib/postgis-1.5 folder.

The rest of this chapter goes into detail each of the above installation steps.

2.2 Requirements

PostGIS has the following requirements for building and usage:

Required

- PostgreSQL 8.3 - 9.1. A complete installation of PostgreSQL (including server headers) is required. PostgreSQL is available from http://www.postgresql.org .

Note
Although people have been successful compiling this version with PostgreSQL 9.2, we do not support nor test 9.2 on this version of PostGIS so use with PostgreSQL 9.2 at your own risk.

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

- GNU C compiler (`gcc`). Some other ANSI C compilers can be used to compile PostGIS, but we find far fewer problems when compiling with `gcc`.
- GNU Make (`gmake` or `make`). For many systems, GNU `make` is the default version of make. Check the version by invoking `make -v`. Other versions of `make` may not process the PostGIS `Makefile` properly.
- Proj4 reprojection library, version 4.6.0 or greater. The Proj4 library is used to provide coordinate reprojection support within PostGIS. Proj4 is available for download from http://trac.osgeo.org/proj/ .
- GEOS geometry library, version 3.1.1 or greater, but GEOS 3.2 is recommended. Without GEOS 3.2, you will be missing some major enhancements with handling of topological exceptions and improvements to ST_Buffer that allow beveling and mitre and much faster buffering. The GEOS library is used to provide geometry tests (ST_Touches(), ST_Contains(), ST_Intersects()) and operations (ST_Buffer(), ST_Union(),ST_Intersection() ST_Difference()) within PostGIS. GEOS is available for download from http://trac.osgeo.org/geos/ .
- LibXML2, version 2.5.x or higher. LibXML2 is currently used in some imports functions (ST_GeomFromGML and ST_GeomFromKI LibXML2 is available for download from http://xmlsoft.org/downloads.html.

Optional

- GTK (requires GTK+2.0) to compile the shp2pgsql-gui shape file loader. http://www.gtk.org/ .
- CUnit (`CUnit`). This is needed for regression tests. http://cunit.sourceforge.net/
- Apache Ant (`ant`) is required for building any of the drivers under the `java` directory. Ant is available from http://ant.apache.org .
- DocBook (`xsltproc`) is required for building the documentation. Docbook is available from http://www.docbook.org/ .
- DBLatex (`dblatex`) is required for building the documentation in PDF format. DBLatex is available from http://dblatex.sourceforge.r .
- ImageMagick (`convert`) is required to generate the images used in the documentation. ImageMagick is available from http://www.imagemagick.org/ .

2.3 Getting the Source

Retrieve the PostGIS source archive from the downloads website http://postgis.net/stuff/postgis-1.5.9SVN.tar.gz

```
wget http://postgis.net/stuff/postgis-1.5.9SVN.tar.gz
tar -xvzf postgis-1.5.9SVN.tar.gz
```

This will create a directory called `postgis-1.5.9SVN` in the current working directory.

Alternatively, checkout the source from the svn repository http://svn.osgeo.org/postgis/trunk/1.5 .

```
svn checkout http://svn.osgeo.org/postgis/branches/1.5 postgis-1.5.9SVN
```

Change into the newly created `postgis-1.5.9SVN` directory to continue the installation.

2.4 Installation

Note
Many OS systems now include pre-built packages for PostgreSQL/PostGIS. In many cases compilation is only necessary if you want the most bleeding edge versions or you are a package maintainer.

The PostGIS module is an extension to the PostgreSQL backend server. As such, PostGIS 1.5.9SVN *requires* full PostgreSQL server headers access in order to compile. It can be built against PostgreSQL versions 8.3 or higher. Earlier versions of PostgreSQL are *not* supported.

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

Note
For GEOS functionality, when you install PostgresSQL you may need to explicitly link PostgreSQL against the standard C++ library:

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

This is a workaround for bogus C++ exceptions interaction with older development tools. If you experience weird problems (backend unexpectedly closed or similar things) try this trick. This will require recompiling your PostgreSQL from scratch, of course.

The following steps outline the configuration and compilation of the PostGIS source. They are written for Linux users and will not work on Windows or Mac.

2.4.1 Configuration

As with most linux installations, the first step is to generate the Makefile that will be used to build the source code. This is done by running the shell script

./configure

With no additional parameters, this command will attempt to automatically locate the required components and libraries needed to build the PostGIS source code on your system. Although this is the most common usage of **./configure**, the script accepts several parameters for those who have the required libraries and programs in non-standard locations.

The following list shows only the most commonly used parameters. For a complete list, use the **--help** or **--help=short** parameters.

--prefix=*PREFIX* This is the location the PostGIS libraries and SQL scripts will be installed to. By default, this location is the same as the detected PostgreSQL installation.

> **Caution**
> This paramater is currently broken, as the package will only install into the PostgreSQL installation directory. Visit http://trac.osgeo.org/postgis/ticket/160 to track this bug.

--with-pgconfig=*FILE* PostgreSQL provides a utility called **pg_config** to enable extensions like PostGIS to locate the PostgreSQL installation directory. Use this parameter (**--with-pgconfig=/path/to/pg_config**) to manually specify a particular PostgreSQL installation that PostGIS will build against.

--with-geosconfig=*FILE* GEOS, a required geometry library, provides a utility called **geos-config** to enable software installations to locate the GEOS installation directory. Use this parameter (**--with-geosconfig=/path/to/geos-config**) to manually specify a particular GEOS installation that PostGIS will build against.

--with-projdir=*DIR* Proj4 is a reprojection library required by PostGIS. Use this parameter (**--with-projdir=/path/to/projdir**) to manually specify a particular Proj4 installation directory that PostGIS will build against.

--with-gui Compile the data import GUI (requires GTK+2.0). This will create shp2pgsql-gui graphical interface to shp2pgsql.

Note

If you obtained PostGIS from the SVN repository , the first step is really to run the script

./autogen.sh

This script will generate the **configure** script that in turn is used to customize the intallation of PostGIS.

If you instead obtained PostGIS as a tarball, running **./autogen.sh** is not necessary as **configure** has already been generated.

2.4.2 Building

Once the Makefile has been generated, building PostGIS is as simple as running

make

The last line of the output should be "`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 is also packaged in the tar.gz distribution in the doc folder so no need to make comments if installing from the tar ball.

make comments

2.4.3 Testing

If you wish to test the PostGIS build, run

make check

The above command will run through various checks and regression tests using the generated library against an actual PostgreSQL database.

Note

If you configured PostGIS using non-standard PostgreSQL, GEOS, or Proj4 locations, you may need to add their library locations to the LD_LIBRARY_PATH environment variable.

Caution

Currently, the **make check** relies on the `PATH` and `PGPORT` environment variables when performing the checks - it does *not* use the PostgreSQL version that may have been specified using the configuration paramter **--with-pgconfig**. So make sure to modify your PATH to match the detected PostgreSQL installation during configuration or be prepared to deal with the impending headaches. Visit http://trac.osgeo.org/postgis/ticket/186 to track this bug.

If successful, the output of the test should be similiar to the following:

```
     CUnit - A Unit testing framework for C - Version 2.1-0
   http://cunit.sourceforge.net/


Suite: PostGIS Computational Geometry Suite
  Test: test_lw_segment_side() ... passed
  Test: test_lw_segment_intersects() ... passed
  Test: test_lwline_crossing_short_lines() ... passed
  Test: test_lwline_crossing_long_lines() ... passed
  Test: test_lwpoint_set_ordinate() ... passed
  Test: test_lwpoint_get_ordinate() ... passed
  Test: test_lwpoint_interpolate() ... passed
  Test: test_lwline_clip() ... passed
  Test: test_lwline_clip_big() ... passed
  Test: test_lwmline_clip() ... passed
  Test: test_geohash_point() ... passed
  Test: test_geohash_precision() ... passed
  Test: test_geohash() ... passed
Suite: PostGIS Measures Suite
  Test: test_mindistance2d_recursive_tolerance() ... passed

--Run Summary: Type      Total     Ran  Passed  Failed
         suites        2       2     n/a       0
         tests        14      14      14       0
         asserts      84      84      84       0


Creating spatial db postgis_reg
TMPDIR is /tmp/pgis_reg_15328

 PostgreSQL 8.3.7 on i686-pc-linux-gnu, compiled by GCC gcc (GCC) 4.1.2 20080704 (Red Hat  ↩
     4.1.2-44)
 Postgis 1.4.0SVN - 2009-05-25 20:21:55
   GEOS: 3.1.0-CAPI-1.5.0
   PROJ: Rel. 4.6.1, 21 August 2008

Running tests

 loader/Point............. ok
 loader/PointM............. ok
 loader/PointZ............. ok
 loader/MultiPoint............. ok
 loader/MultiPointM............. ok
 loader/MultiPointZ............. ok
 loader/Arc............. ok
 loader/ArcM............. ok
 loader/ArcZ.......... ok
 loader/Polygon............. ok
 loader/PolygonM............. ok
 loader/PolygonZ............. ok
 regress. ok
 regress_index. ok
 regress_index_nulls. ok
 lwgeom_regress. ok
 regress_lrs. ok
 removepoint. ok
 setpoint. ok
 simplify. ok
 snaptogrid. ok
 affine. ok
 wkt. ok
 measures. ok
 long_xact. ok
 ctors. ok
 sql-mm-serialize. ok
 sql-mm-circularstring. ok
 sql-mm-compoundcurve. ok
 sql-mm-curvepoly. ok
 sql-mm-general. ok
 sql-mm-multicurve. ok
 sql-mm-multisurface. ok
 geojson. ok
 gml. ok
 svg. ok
 kml. ok
 regress_ogc. ok
 regress_bdpoly. ok
 regress_proj. ok
 regress_ogc_cover. ok
 regress_ogc_prep. ok

Run tests: 42
Failed: 0
```

2.4.4 Installation

To install PostGIS, type

make install

This will copy the PostGIS installation files into their appropriate subdirectory specified by the **--prefix** configuration parameter. In particular:

- The loader and dumper binaries are installed in `[prefix]/bin`.
- The SQL files, such as `postgis.sql`, are installed in `[prefix]/share/contrib`.
- The PostGIS libraries are installed in `[prefix]/lib`.

If you previously ran the **make comments** command to generate the `postgis_comments.sql` file, install the sql file by running

make comments-install

Note
`postgis_comments.sql` was separated from the typical build and installation targets since with it comes the extra dependency of **xsltproc**.

2.5 Create a spatially-enabled database

The first step in creating a PostGIS database is to create a simple PostgreSQL database.

createdb [yourdatabase]

Many of the PostGIS functions are written in the PL/pgSQL procedural language. As such, the next step to create a PostGIS database is to enable the PL/pgSQL language in your new database. This is accomplish by the command

createlang plpgsql [yourdatabase]

Now load the PostGIS object and function definitions into your database by loading the `postgis.sql` definitions file (located in `[prefix]/share/contrib` as specified during the configuration step).

psql -d [yourdatabase] -f postgis.sql

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.

psql -d [yourdatabase] -f spatial_ref_sys.sql

If you wish to add comments to the PostGIS functions, the final step is to load the `postgis_comments.sql` into your spatial database. The comments can be viewed by simply typing **\dd [function_name]** from a **psql** terminal window.

psql -d [yourdatabase] -f postgis_comments.sql

2.6 Create a spatially-enabled database from a template

Some packaged distributions of PostGIS (in particular the Win32 installers for PostGIS >= 1.1.5) load the PostGIS functions into a template database called `template_postgis`. If the `template_postgis` database exists in your PostgreSQL installation then it is possible for users and/or applications to create spatially-enabled databases using a single command. Note that in both cases, the database user must have been granted the privilege to create new databases.

From the shell:

```
# createdb -T template_postgis my_spatial_db
```

From SQL:

```
postgres=# CREATE DATABASE my_spatial_db TEMPLATE=template_postgis
```

2.7 Upgrading

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 an 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 an HARD UPGRADE.

2.7.1 Soft upgrade

After compiling you should find several `postgis_upgrade*.sql` files. Install the one for your version of PostGIS. For example `postgis_upgrade_13_to_15.sql` should be used if you are upgrading from postgis 1.3 to 1.5.

```
$ psql -f postgis_upgrade_13_to_15.sql -d your_spatial_database
```

If a soft upgrade is not possible the script will abort and you will be warned about HARD UPGRADE being required, so do not hesitate to try a soft upgrade first.

Note

If you can't find the `postgis_upgrade*.sql` files you are probably using a version prior to 1.1 and must generate that file by yourself. This is done with the following command:

```
$ utils/postgis_proc_upgrade.pl postgis.sql > postgis_upgrade.sql
```

2.7.2 Hard upgrade

By HARD UPGRADE we intend full dump/reload of postgis-enabled databases. You need an 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.

PostGIS provides an utility script to restore a dump produced with the pg_dump -Fc command. It is experimental so redirecting its output to a file will help in case of problems. The procedure is as follow:

Create a "custom-format" dump of the database you want to upgrade (let's call it "olddb")

```
$ pg_dump -Fc olddb > olddb.dump
```

Restore the dump contextually upgrading postgis into a new database. The new database doesn't have to exist. postgis_restore accepts createdb parameters after the dump file name, and that can for instance be used if you are using a non-default character encoding for your database. Let's call it "newdb", with UNICODE as the character encoding:

```
$ sh utils/postgis_restore.pl postgis.sql newdb olddb.dump -E=UNICODE > restore.log
```

Check that all restored dump objects really had to be restored from dump and do not conflict with the ones defined in postgis.sql

```
$ grep ^KEEPING restore.log | less
```

If upgrading from PostgreSQL < 8.0 to >= 8.0 you might want to drop the attrelid, varattnum and stats columns in the geometry_columns table, which are no-more needed. Keeping them won't hurt. DROPPING THEM WHEN REALLY NEEDED WILL DO HURT !

```
$ psql newdb -c "ALTER TABLE geometry_columns DROP attrelid"
$ psql newdb -c "ALTER TABLE geometry_columns DROP varattnum"
$ psql newdb -c "ALTER TABLE geometry_columns DROP stats"
```

spatial_ref_sys table is restore from the dump, to ensure your custom additions are kept, but the distributed one might contain modification so you should backup your entries, drop the table and source the new one. If you did make additions we assume you know how to backup them before upgrading the table. Replace of it with the new one is done like this:

```
$ psql newdb
newdb=> truncate spatial_ref_sys;
TRUNCATE
newdb=> \i spatial_ref_sys.sql
```

2.8 Common Problems

There are several things to check when your installation or upgrade doesn't go as you expected.

1. Check that you you have installed PostgreSQL 8.1 or newer, and that you are compiling against the same version of the PostgreSQL source as the version of PostgreSQL that is running. Mix-ups can occur when your (Linux) distribution has already installed PostgreSQL, or you have otherwise installed PostgreSQL before and forgotten about it. PostGIS will only work with PostgreSQL 8.1 or newer, and strange, unexpected error messages will result if you use an older version. To check the version of PostgreSQL which is running, connect to the database using psql and run this query:

   ```
   SELECT version();
   ```

   If you are running an RPM based distribution, you can check for the existence of pre-installed packages using the **rpm** command as follows: **rpm -qa | grep postgresql**

Also check that configure has correctly detected the location and version of PostgreSQL, the Proj4 library and the GEOS library.

1. The output from configure is used to generate the `postgis_config.h` file. Check that the `POSTGIS_PGSQL_VERSION`, `POSTGIS_PROJ_VERSION` and `POSTGIS_GEOS_VERSION` variables have been set correctly.

2.9 JDBC

The JDBC extensions provide Java objects corresponding to the internal PostGIS types. These objects can be used to write Java clients which query the PostGIS database and draw or do calculations on the GIS data in PostGIS.

1. Enter the `java/jdbc` sub-directory of the PostGIS distribution.
2. Run the `ant` command. Copy the `postgis.jar` file to wherever you keep your java libraries.

The JDBC extensions require a PostgreSQL JDBC driver to be present in the current CLASSPATH during the build process. If the PostgreSQL JDBC driver is located elsewhere, you may pass the location of the JDBC driver JAR separately using the -D parameter like this:

```
# ant -Dclasspath=/path/to/postgresql-jdbc.jar
```

PostgreSQL JDBC drivers can be downloaded from http://jdbc.postgresql.org .

2.10 Loader/Dumper

The data loader and dumper are built and installed automatically as part of the PostGIS build. To build and install them manually:

```
# cd postgis-1.5.9SVN/loader
# make
# make install
```

The loader is called `shp2pgsql` and converts ESRI Shape files into SQL suitable for loading in PostGIS/PostgreSQL. The dumper is called `pgsql2shp` and converts PostGIS tables (or queries) into ESRI Shape files. For more verbose documentation, see the online help, and the manual pages.

Chapter 3

Frequently Asked Questions

1. *I'm running PostgreSQL 9.0 and I can no longer read/view geometries in OpenJump, Safe FME, and some other tools?*

   In PostgreSQL 9.0+, the default encoding for bytea data has been changed to hex and older JDBC drivers still assume escape format. This has affected some applications such as Java applications using older JDBC drivers or .NET applications that use the older npgsql driver that expect the old behavior of ST_AsBinary. There are two approaches to getting this to work again.You can upgrade your JDBC driver to the latest PostgreSQL 9.0 version which you can get from http://jdbc.postgresql.org/download.htmlIf you are running a .NET app, you can use Npgsql 2.0.11 or higher which you can download from http://pgfoundry.org/frs/?group_id=1000140 and as described on Francisco Figueiredo's NpgSQL 2.0.11 released blog entryIf upgrading your PostgreSQL driver is not an option, then you can set the default back to the old behavior with the following change:

   ```
   ALTER DATABASE mypostgisdb SET bytea_output='escape';
   ```

2. *I tried to use PgAdmin to view my geometry column and it is blank, what gives?*

   PgAdmin doesn't show anything for large geometries. The best ways to verify you do have day in your geometry columns are?

   ```
   -- this should return no records if all your geom fields are filled in
   SELECT somefield FROM mytable WHERE geom IS NULL;
   ```

   ```
   -- To tell just how large your geometry is do a query of the form
   --which will tell you the most number of points you have in any of your geometry ↩
       columns
   SELECT MAX(ST_NPoints(geom)) FROM sometable;
   ```

3. *What kind of geometric objects can I store?*

   You can store point, line, polygon, multipoint, multiline, multipolygon, and geometrycollections. These are specified in the Open GIS Well Known Text Format (with XYZ,XYM,XYZM extensions). There are two data types currently supported. The standard OGC geometry data type which uses a planar coordinate system for measurement and the geography data type which uses a geodetic coordinate system. Only WGS 84 long lat (SRID:4326) is supported by the geography data type.

4. *I'm all confused. Which data store should I use geometry or geography?*

   Short Answer: geography is a new data type that supports long range distances measurements, but most computations on it are currently slower than they are on geometry. If you use geography -- you don't need to learn much about planar coordinate systems. Geography is generally best if all you care about is measuring distances and lengths and you have data from all over the world. Geometry data type is an older data type that has many more functions supporting it, enjoys greater support from third party tools, and operations on it are generally faster -- sometimes as much as 10 fold faster for larger geometries. Geometry is best if you are pretty comfortable with spatial reference systems or you are dealing with localized data where all your data fits in a single spatial reference system (SRID), or you need to do a lot of spatial processing. Note: It is fairly easy to do one-off conversions between the two types to gain the benefits of each. Refer to

Section 8.8 to see what is currently supported and what is not. Long Answer: Refer to our more lengthy discussion in the Section 4.2.2 and function type matrix.

5. *I have more intense questions about geography, such as how big of a geographic region can I stuff in a geography column and still get reasonable answers. Are there limitations such as poles, everything in the field must fit in a hemisphere (like SQL Server 2008 has), speed etc?*

   Your questions are too deep and complex to be adequately answered in this section. Please refer to our Section 4.2.3 .

6. *How do I insert a GIS object into the database?*

   First, you need to create a table with a column of type "geometry" or "geography" to hold your GIS data. Storing geography type data is a little different than storing geometry. Refer to Section 4.2.1 for details on storing geography. For geometry: Connect to your database with `psql` and try the following SQL:

   ```
   CREATE TABLE gtest ( ID int4, NAME varchar(20) );
   SELECT AddGeometryColumn('', 'gtest','geom',-1,'LINESTRING',2);
   ```

   If the geometry column addition fails, you probably have not loaded the PostGIS functions and objects into this database. See the Section 2.4.Then, you can insert a geometry into the table using a SQL insert statement. The GIS object itself is formatted using the OpenGIS Consortium "well-known text" format:

   ```
   INSERT INTO gtest (ID, NAME, GEOM)
   VALUES (
     1,
     'First Geometry',
     ST_GeomFromText('LINESTRING(2 3,4 5,6 5,7 8)', -1)
   );
   ```

   For more information about other GIS objects, see the object reference.To view your GIS data in the table:

   ```
   SELECT id, name, ST_AsText(geom) AS geom FROM gtest;
   ```

   The return value should look something like this:

   ```
    id | name           | geom
   ----+----------------+-----------------------------
     1 | First Geometry | LINESTRING(2 3,4 5,6 5,7 8)
   (1 row)
   ```

7. *How do I construct a spatial query?*

   The same way you construct any other database query, as an SQL combination of return values, functions, and boolean tests.For spatial queries, there are two issues that are important to keep in mind while constructing your query: is there a spatial index you can make use of; and, are you doing expensive calculations on a large number of geometries.In general, you will want to use the "intersects operator" (&&) which tests whether the bounding boxes of features intersect. The reason the && operator is useful is because if a spatial index is available to speed up the test, the && operator will make use of this. This can make queries much much faster.You will also make use of spatial functions, such as Distance(), ST_Intersects(), ST_Contains() and ST_Within(), among others, to narrow down the results of your search. Most spatial queries include both an indexed test and a spatial function test. The index test serves to limit the number of return tuples to only tuples that *might* meet the condition of interest. The spatial functions are then use to test the condition exactly.

   ```
   SELECT id, the_geom
   FROM thetable
   WHERE
     ST_Contains(the_geom,'POLYGON((0 0, 0 10, 10 10, 10 0, 0 0))');
   ```

8. *How do I speed up spatial queries on large tables?*

   Fast queries on large tables is the *raison d'etre* of spatial databases (along with transaction support) so having a good index is important.To build a spatial index on a table with a `geometry` column, use the "CREATE INDEX" function as follows:

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

The "USING GIST" option tells the server to use a GiST (Generalized Search Tree) index.

Note
GiST indexes are assumed to be lossy. Lossy indexes uses a proxy object (in the spatial case, a bounding box) for building the index.

You should also ensure that the PostgreSQL query planner has enough information about your index to make rational decisions about when to use it. To do this, you have to "gather statistics" on your geometry tables.For PostgreSQL 8.0.x and greater, just run the **VACUUM ANALYZE** command.For PostgreSQL 7.4.x and below, run the **SELECT UPDATE_GEOMETRY_STATS()** command.

9. *Why aren't PostgreSQL R-Tree indexes supported?*

   Early versions of PostGIS used the PostgreSQL R-Tree indexes. However, PostgreSQL R-Trees have been completely discarded since version 0.6, and spatial indexing is provided with an R-Tree-over-GiST scheme.Our tests have shown search speed for native R-Tree and GiST to be comparable. Native PostgreSQL R-Trees have two limitations which make them undesirable for use with GIS features (note that these limitations are due to the current PostgreSQL native R-Tree implementation, not the R-Tree concept in general):

   - R-Tree indexes in PostgreSQL cannot handle features which are larger than 8K in size. GiST indexes can, using the "lossy" trick of substituting the bounding box for the feature itself.
   - R-Tree indexes in PostgreSQL are not "null safe", so building an index on a geometry column which contains null geometries will fail.

10. *Why should I use the `AddGeometryColumn()` function and all the other OpenGIS stuff?*

    If you do not want to use the OpenGIS support functions, you do not have to. Simply create tables as in older versions, defining your geometry columns in the CREATE statement. All your geometries will have SRIDs of -1, and the OpenGIS meta-data tables will *not* be filled in properly. However, this will cause most applications based on PostGIS to fail, and it is generally suggested that you do use `AddGeometryColumn()` to create geometry tables.MapServer is one application which makes use of the `geometry_columns` meta-data. Specifically, MapServer can use the SRID of the geometry column to do on-the-fly reprojection of features into the correct map projection.

11. *What is the best way to find all objects within a radius of another object?*

    To use the database most efficiently, it is best to do radius queries which combine the radius test with a bounding box test: the bounding box test uses the spatial index, giving fast access to a subset of data which the radius test is then applied to.The `ST_DWithin(geometry, geometry, distance)` function is a handy way of performing an indexed distance search. It works by creating a search rectangle large enough to enclose the distance radius, then performing an exact distance search on the indexed subset of results.For example, to find all objects with 100 meters of POINT(1000 1000) the following query would work well:

    ```
    SELECT * FROM geotable
    WHERE ST_DWithin(geocolumn, 'POINT(1000 1000)', 100.0);
    ```

12. *How do I perform a coordinate reprojection as part of a query?*

    To perform a reprojection, both the source and destination coordinate systems must be defined in the SPATIAL_REF_SYS table, and the geometries being reprojected must already have an SRID set on them. Once that is done, a reprojection is as simple as referring to the desired destination SRID. The below projects a geometry to NAD 83 long lat. The below will only work if the srid of the_geom is not -1 (not undefined spatial ref)

    ```
    SELECT ST_Transform(the_geom,4269) FROM geotable;
    ```

13. *I did an ST_AsEWKT and ST_AsText on my rather large geometry and it returned blank field. What gives?*

    You are probably using PgAdmin or some other tool that doesn't output large text. If your geometry is big enough, it will appear blank in these tools. Use PSQL if you really need to see it or output it in WKT.

    ```
            --To check number of geometries are really blank
            SELECT count(gid) FROM geotable WHERE the_geom IS NULL;
    ```

14. *When I do an ST_Intersects, it says my two geometries don't intersect when I KNOW THEY DO. What gives?*

    This generally happens in two common cases. Your geometry is invalid -- check ST_IsValid or you are assuming they intersect because ST_AsText truncates the numbers and you have lots of decimals after it is not showing you.

Chapter 4

Using PostGIS: Data Management and Queries

4.1 GIS Objects

The GIS objects supported by PostGIS are a superset of the "Simple Features" defined by the OpenGIS Consortium (OGC). As of version 0.9, PostGIS supports all the objects and functions specified in the OGC "Simple Features for SQL" specification.

PostGIS extends the standard with support for 3DZ,3DM and 4D coordinates.

4.1.1 OpenGIS WKB and WKT

The OpenGIS specification defines two standard ways of expressing spatial objects: the Well-Known Text (WKT) form and the Well-Known Binary (WKB) form. Both WKT and WKB include information about the type of the object and the coordinates which form the object.

Examples of the text representations (WKT) of the spatial objects of the features are as follows:

- POINT(0 0)
- LINESTRING(0 0,1 1,1 2)
- 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)
- 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))

The OpenGIS specification also requires that the internal storage format of spatial objects include a spatial referencing system identifier (SRID). The SRID is required when creating spatial objects for insertion into the database.

Input/Output of these formats are available using the following interfaces:

```
bytea WKB = ST_AsBinary(geometry);
text WKT = ST_AsText(geometry);
geometry = ST_GeomFromWKB(bytea WKB, SRID);
geometry = ST_GeometryFromText(text WKT, SRID);
```

For example, a valid insert statement to create and insert an OGC spatial object would be:

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

4.1.2 PostGIS EWKB, EWKT and Canonical Forms

OGC formats only support 2d geometries, and the associated SRID is *never* embedded in the input/output representations.

PostGIS extended formats are currently superset of OGC one (every valid WKB/WKT is a valid EWKB/EWKT) but this might vary in the future, specifically if OGC comes out with a new format conflicting with our extensions. Thus you SHOULD NOT rely on this feature!

PostGIS EWKB/EWKT add 3dm,3dz,4d coordinates support and embedded SRID information.

Examples of the text representations (EWKT) of the extended spatial objects of the features are as follows:

- 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))

Input/Output of these formats are available using the following interfaces:

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

For example, a valid insert statement to create and insert a PostGIS spatial object would be:

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

The "canonical forms" of a PostgreSQL type are the representations you get with a simple query (without any function call) and the one which is guaranteed to be accepted with a simple insert, update or copy. For the postgis 'geometry' type these are:

```
- Output
  - binary: EWKB
  ascii: HEXEWKB (EWKB in hex form)
- Input
  - binary: EWKB
  ascii: HEXEWKB|EWKT
```

For example this statement reads EWKT and returns HEXEWKB in the process of canonical ascii input/output:

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

geometry
----------------------------------------------------
01010000200400000000000000000000000000000000000000
(1 row)
```

4.1.3 SQL-MM Part 3

The SQL Multimedia Applications Spatial specification extends the simple features for SQL spec by defining a number of circularly interpolated curves.

The SQL-MM definitions include 3dm, 3dz and 4d coordinates, but do not allow the embedding of SRID information.

The well-known text extensions are not yet fully supported. Examples of some simple curved geometries are shown below:

- CIRCULARSTRING(0 0, 1 1, 1 0)

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

  The CIRCULARSTRING is the basic curve type, similar to a LINESTRING in the linear world. A single segment required three points, the start and end points (first and third) and any other point on the arc. The exception to this is for a closed circle, where the start and end points are the same. In this case the second point MUST be the center of the arc, ie the opposite side of the circle. To chain arcs together, the last point of the previous arc becomes the first point of the next arc, just like in LINESTRING. This means that a valid circular string must have an odd number of points greated than 1.

- COMPOUNDCURVE(CIRCULARSTRING(0 0, 1 1, 1 0),(1 0, 0 1))

  A compound curve is a single, continuous curve that has both curved (circular) segments and linear segments. That means that in addition to having well-formed components, the end point of every component (except the last) must be coincident with the start point of the following component.

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

  Example compound curve in a curve polygon: 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))

  A CURVEPOLYGON is just like a polygon, with an outer ring and zero or more inner rings. The difference is that a ring can take the form of a circular string, linear string or compound string.

  As of PostGIS 1.4 PostGIS supports compound curves in a curve polygon.

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

  The MULTICURVE is a collection of curves, which can include linear strings, circular strings or compound strings.

- 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)))

  This is a collection of surfaces, which can be (linear) polygons or curve polygons.

Note
PostGIS prior to 1.4 does not support compound curves in a curve polygon, but PostGIS 1.4 and above do support the use of Compound Curves in a Curve Polygon.

Note
All floating point comparisons within the SQL-MM implementation are performed to a specified tolerance, currently 1E-8.

4.2 PostGIS Geography Type

The geography type provides native support for spatial features represented on "geographic" coordinates (sometimes called "geodetic" coordinates, or "lat/lon", or "lon/lat"). Geographic coordinates are spherical coordinates expressed in angular units (degrees).

The basis for the PostGIS geometry type is a plane. The shortest path between two points on the plane is a straight line. That means calculations on geometries (areas, distances, lengths, intersections, etc) can be calculated using cartesian mathematics and straight line vectors.

The basis for the PostGIS geographic type is a sphere. The shortest path between two points on the sphere is a great circle arc. That means that calculations on geographies (areas, distances, lengths, intersections, etc) must be calculated on the sphere, using more complicated mathematics. For more accurate measurements, the calculations must take the actual spheroidal shape of the world into account, and the mathematics becomes very complicated indeed.

Because the underlying mathematics is much more complicated, there are fewer functions defined for the geography type than for the geometry type. Over time, as new algorithms are added, the capabilities of the geography type will expand.

One restriction is that it only supports WGS 84 long lat (SRID:4326). It uses a new data type called geography. None of the GEOS functions support this new type. As a workaround one can convert back and forth between geometry and geography types.

The new geography type uses the PostgreSQL 8.3+ typmod definition format so that a table with a geography field can be added in a single step. All the standard OGC formats except for curves are supported.

4.2.1 Geography Basics

The geography type only supports the simplest of simple features. Standard geometry type data will autocast to geography if it is of SRID 4326. You can also use the EWKT and EWKB conventions to insert data.

- POINT: Creating a table with 2d point geometry:

  ```
  CREATE TABLE testgeog(gid serial PRIMARY KEY, the_geog geography(POINT,4326) );
  ```

  Creating a table with z coordinate point

  ```
  CREATE TABLE testgeog(gid serial PRIMARY KEY, the_geog geography(POINTZ,4326) );
  ```

- LINESTRING
- POLYGON
- MULTIPOINT
- MULTILINESTRING
- MULTIPOLYGON
- GEOMETRYCOLLECTION

The new geography fields don't get registered in the geometry_columns. They get registered in a new view called geography_columns which is a view against the system catalogs so is always automatically kept up to date without need for an AddGeom... like function.

Now, check the "geography_columns" view and see that your table is listed.

You can create a new table with a GEOGRAPHY column using the CREATE TABLE syntax. Unlike GEOMETRY, there is no need to run a separate AddGeometryColumns() process to register the column in metadata.

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

Note that the location column has type GEOGRAPHY and that geography type supports two optional modifier: a type modifier that restricts the kind of shapes and dimensions allowed in the column; an SRID modifier that restricts the coordinate reference identifier to a particular number.

Allowable values for the type modifier are: POINT, LINESTRING, POLYGON, MULTIPOINT, MULTILINESTRING, MULTIPOLYGON. The modifier also supports dimensionality restrictions through suffixes: Z, M and ZM. So, for example a modifier of 'LINESTRINGM' would only allow line strings with three dimensions in, and would treat the third dimension as a measure. Similarly, 'POINTZM' would expect four dimensional data.

The SRID modifier is currently of limited use: only 4326 (WGS84) is allowed as a value. If you do not specify an SRID, the a value 0 (undefined spheroid) will be used, and all calculations will proceed using WGS84 anyways.

In the future, alternate SRIDs will allow calculations on spheroids other than WGS84.

Once you have created your table, you can see it in the GEOGRAPHY_COLUMNS table:

```
-- See the contents of the metadata view
SELECT * FROM geography_columns;
```

You can insert data into the table the same as you would if it was using a GEOMETRY column:

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

Creating an index works the same as GEOMETRY. PostGIS will note that the column type is GEOGRAPHY and create an appropriate sphere-based index instead of the usual planar index used for GEOMETRY.

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

Query and measurement functions use units of meters. So distance parameters should be expressed in meters, and return values should be expected in meters (or square meters for areas).

```
-- Show a distance query and note, London is outside the 1000km tolerance
  SELECT name FROM global_points WHERE ST_DWithin(location, ST_GeographyFromText('SRID ↩
      =4326;POINT(-110 29)'), 1000000);
```

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

```
-- Distance calculation using GEOGRAPHY (122.2km)
  SELECT ST_Distance('LINESTRING(-122.33 47.606, 0.0 51.5)'::geography, 'POINT(-21.96  ↩
      64.15)':: geography);
```

```
-- Distance calculation using GEOMETRY (13.3 "degrees")
  SELECT ST_Distance('LINESTRING(-122.33 47.606, 0.0 51.5)'::geometry, 'POINT(-21.96 64.15) ↩
      ':: geometry);
```

The GEOGRAPHY type calculates the true shortest distance over the sphere between Reykjavik and the great circle flight path between Seattle and London.

Great Circle mapper The GEOMETRY type calculates a meaningless cartesian distance between Reykjavik and the straight line path from Seattle to London plotted on a flat map of the world. The nominal units of the result might be called "degrees", but the result doesn't correspond to any true angular difference between the points, so even calling them "degrees" is inaccurate.

4.2.2 When to use Geography Data type over Geometry data type

The new GEOGRAPHY 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.

The type you choose should be conditioned on the expected working area of the application you are building. Will your data span the globe or a large continental area, or is it local to a state, county or municipality?

- If your data is contained in a small area, you might find that choosing an appropriate projection and using GEOMETRY is the best solution, in terms of performance and functionality available.
- If your data is global or covers a continental region, you may find that GEOGRAPHY allows you to build a system without having to worry about projection details. You store your data in longitude/latitude, and use the functions that have been defined on GEOGRAPHY.
- If you don't understand projections, and you don't want to learn about them, and you're prepared to accept the limitations in functionality available in GEOGRAPHY, then it might be easier for you to use GEOGRAPHY than GEOMETRY. Simply load your data up as longitude/latitude and go from there.

Refer to Section 8.8 for compare between what is supported for Geography vs. Geometry. For a brief listing and description of Geography functions, refer to Section 8.3

4.2.3 Geography Advanced FAQ

1. *Do you calculate on the sphere or the spheroid?*

   By default, all distance and area calculations are done on the spheroid. You should find that the results of calculations in local areas match up will with local planar results in good local projections. Over larger areas, the spheroidal calculations will be more accurate than any calculation done on a projected plane. All the geography functions have the option of using a sphere calculation, by setting a final boolean parameter to 'FALSE'. This will somewhat speed up calculations, particularly for cases where the geometries are very simple.

2. *What about the date-line and the poles?*

   All the calculations have no conception of date-line or poles, the coordinates are spherical (longitude/latitude) so a shape that crosses the dateline is, from a calculation point of view, no different from any other shape.

3. *What is the longest arc you can process?*

   We use great circle arcs as the "interpolation line" between two points. That means any two points are actually joined up two ways, depending on which direction you travel along the great circle. All our code assumes that the points are joined by the *shorter* of the two paths along the great circle. As a consequence, shapes that have arcs of more than 180 degrees will not be correctly modelled.

4. *Why is it so slow to calculate the area of Europe / Russia / insert big geographic region here ?*

   Because the polygon is so darned huge! Big areas are bad for two reasons: their bounds are huge, so the index tends to pull the feature no matter what query you run; the number of vertices is huge, and tests (distance, containment) have to traverse the vertex list at least once and sometimes N times (with N being the number of vertices in the other candidate feature). 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. Just because you *can* store all of Europe in one polygon doesn't mean you *should*.

4.3 Using OpenGIS Standards

The OpenGIS "Simple Features Specification for SQL" defines standard GIS object types, the functions required to manipulate them, and a set of meta-data tables. In order to ensure that meta-data remain consistent, operations such as creating and removing a spatial column are carried out through special procedures defined by OpenGIS.

There are two OpenGIS meta-data tables: `SPATIAL_REF_SYS` and `GEOMETRY_COLUMNS`. The `SPATIAL_REF_SYS` table holds the numeric IDs and textual descriptions of coordinate systems used in the spatial database.

4.3.1 The SPATIAL_REF_SYS Table and Spatial Reference Systems

The spatial_ref_sys table is a PostGIS included and OGC compliant database table that lists over 3000 known spatial reference systems and details needed to transform/reproject between them.

Although the PostGIS spatial_ref_sys table contains over 3000 of the more commonly used spatial reference system definitions that can be handled by the proj library, it does not contain all known to man and you can even define your own custom projection if you are familiar with proj4 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.

An excellent resource for finding spatial reference systems not defined in the core set is http://spatialreference.org/

Some of the more commonly used spatial reference systems are: 4326 - WGS 84 Long Lat, 4269 - NAD 83 Long Lat, 3395 - WGS 84 World Mercator, 2163 - US National Atlas Equal Area, Spatial reference systems for each NAD 83, WGS 84 UTM zone - UTM zones are one of the most ideal for measurement, but only cover 6-degree regions.

Various US state plane spatial reference systems (meter or feet based) - usually one or 2 exists per US state. Most of the meter ones are in the core set, but many of the feet based ones or ESRI created ones you will need to pull from spatialreference.org.

For details on determining which UTM zone to use for your area of interest, check out the utmzone PostGIS plpgsql helper function.

The SPATIAL_REF_SYS table definition is as follows:

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

The SPATIAL_REF_SYS columns are as follows:

SRID An integer value that uniquely identifies the Spatial Referencing System (SRS) within the database.

AUTH_NAME The name of the standard or standards body that is being cited for this reference system. For example, "EPSG" would be a valid AUTH_NAME.

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 where the EPSG projection code would go.

SRTEXT The Well-Known Text representation of the Spatial Reference System. An example of a WKT SRS representation is:

```
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 listing of EPSG projection codes and their corresponding WKT representations, see http://www.opengeospatial.org/. For a discussion of WKT in general, see the OpenGIS "Coordinate Transformation Services Implementation Specification" at http://www.opengeospatial.org/standards. For information on the European Petroleum Survey Group (EPSG) and their database of spatial reference systems, see http://www.epsg.org.

PROJ4TEXT PostGIS uses the Proj4 library to provide coordinate transformation capabilities. The PROJ4TEXT column contains the Proj4 coordinate definition string for a particular SRID. For example:

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

For more information about, see the Proj4 web site at http://trac.osgeo.org/proj/. The spatial_ref_sys.sql file contains both SRTEXT and PROJ4TEXT definitions for all EPSG projections.

4.3.2 The GEOMETRY_COLUMNS Table

The GEOMETRY_COLUMNS table definition is as follows:

```
CREATE TABLE geometry_columns (
  f_table_catalog    VARRCHAR(256) NOT NULL,
  f_table_schema     VARCHAR(256) NOT NULL,
  f_table_name        VARCHAR(256) NOT NULL,
  f_geometry_column  VARCHAR(256) NOT NULL,
  coord_dimension    INTEGER NOT NULL,
  srid               INTEGER NOT NULL,
  type               VARCHAR(30) NOT NULL
)
```

The columns are as follows:

F_TABLE_CATALOG, F_TABLE_SCHEMA, F_TABLE_NAME The fully qualified name of the feature table containing the geometry column. Note that the terms "catalog" and "schema" are Oracle-ish. There is not PostgreSQL analogue of "catalog" so that column is left blank -- for "schema" the PostgreSQL schema name is used (public is the default).

F_GEOMETRY_COLUMN The name of the geometry column in the feature table.

COORD_DIMENSION The spatial dimension (2, 3 or 4 dimensional) of the column.

SRID The ID of the spatial reference system used for the coordinate geometry in this table. It is a foreign key reference to the SPATIAL_REF_SYS.

TYPE The type of the spatial object. To restrict the spatial column to a single type, use one of: POINT, LINESTRING, POLYGON, MULTIPOINT, MULTILINESTRING, MULTIPOLYGON, GEOMETRYCOLLECTION or corresponding XYM versions POINTM, LINESTRINGM, POLYGONM, MULTIPOINTM, MULTILINESTRINGM, MULTIPOLYGONM, GEOMETRYCOLLECTIONM. For heterogeneous (mixed-type) collections, you can use "GEOMETRY" as the type.

> **Note**
> This attribute is (probably) not part of the OpenGIS specification, but is required for ensuring type homogeneity.

4.3.3 Creating a Spatial Table

Creating a table with spatial data is done in two stages:

- Create a normal non-spatial table.

  For example: **CREATE TABLE ROADS_GEOM (ID int4, NAME varchar(25))**

- Add a spatial column to the table using the OpenGIS "AddGeometryColumn" function.

  The syntax is:

```
AddGeometryColumn(
  <schema_name>,
  <table_name>,
  <column_name>,
  <srid>,
  <type>,
  <dimension>
)
```

Or, using current schema:

```
AddGeometryColumn(
  <table_name>,
  <column_name>,
  <srid>,
  <type>,
  <dimension>
)
```

Example1: **SELECT AddGeometryColumn('public', 'roads_geom', 'geom', 423, 'LINESTRING', 2)**

Example2: **SELECT AddGeometryColumn('roads_geom', 'geom', 423, 'LINESTRING', 2)**

Here is an example of SQL used to create a table and add a spatial column (assuming that an SRID of 128 exists already):

```
CREATE TABLE parks (
  park_id    INTEGER,
  park_name  VARCHAR,
  park_date  DATE,
  park_type  VARCHAR
);
SELECT AddGeometryColumn('parks', 'park_geom', 128, 'MULTIPOLYGON', 2 );
```

Here is another example, using the generic "geometry" type and the undefined SRID value of -1:

```
CREATE TABLE roads (
  road_id INTEGER,
  road_name VARCHAR
);
SELECT AddGeometryColumn( 'roads', 'roads_geom', -1, 'GEOMETRY', 3 );
```

4.3.4 Manually Registering Geometry Columns in geometry_columns

The AddGeometryColumn() approach creates a geometry column and also registers the new column in the geometry_columns table. If your software utilizes geometry_columns, then any geometry columns you need to query by must be registered in this table. Two of the cases where you want a geometry column to be registered in the geometry_columns table, but you can't use AddGeometryColumn, is in the case of SQL Views and bulk inserts. For these cases, you must register the column in the geometry_columns table manually. Below is a simple script to do that.

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

--To register this table in AddGeometry columns - do the following
INSERT INTO geometry_columns(f_table_catalog, f_table_schema, f_table_name,  ↩
    f_geometry_column, coord_dimension, srid, "type")
SELECT '', 'public', 'vwmytablemercator', 'the_geom', ST_CoordDim(the_geom), ST_SRID( ↩
    the_geom), GeometryType(the_geom)
FROM public.vwmytablemercator LIMIT 1;
```

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

--Create index on new table
CREATE INDEX idx_myschema_myspecialpois_geom_gist
  ON myschema.myspecialpois USING gist(the_geom);

--To manually register this new table's geometry column in geometry_columns
-- we do the same thing as with view
INSERT INTO geometry_columns(f_table_catalog, f_table_schema, f_table_name,  ↩
    f_geometry_column, coord_dimension, srid, "type")
SELECT '', 'myschema', 'myspecialpois', 'the_geom', ST_CoordDim(the_geom), ST_SRID(the_geom ↩
    ), GeometryType(the_geom)
FROM public.myschema.myspecialpois LIMIT 1;
```

4.3.5 Ensuring OpenGIS compliancy of geometries

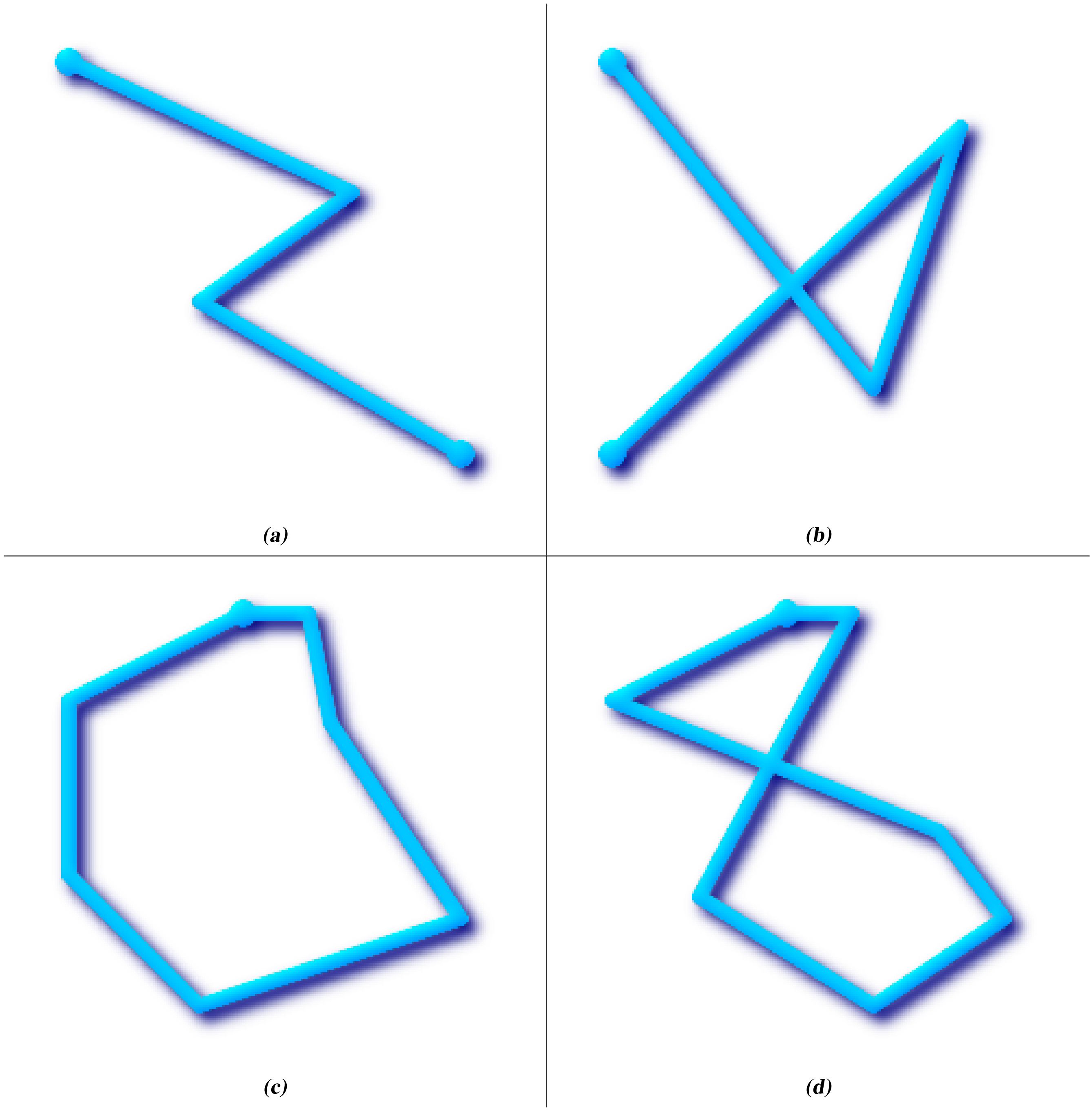
PostGIS is compliant with the Open Geospatial Consortium's (OGC) OpenGIS Specifications. As such, many PostGIS methods require, or more accurately, assume that geometries that are operated on are both simple and valid. for example, it does not make sense to calculate the area of a polygon that has a hole defined outside of the polygon, or to construct a polygon from a non-simple boundary line.

According to the OGC Specifications, a *simple* geometry is one that has no anomalous geometric points, such as self intersection or self tangency and primarily refers to 0 or 1-dimensional geometries (i.e. `[MULTI]POINT, [MULTI]LINESTRING`). Geometry validity, on the other hand, primarily refers to 2-dimensional geometries (i.e. `[MULTI]POLYGON)` and defines the set of assertions that characterizes a valid polygon. The description of each geometric class includes specific conditions that further detail geometric simplicity and validity.

A `POINT` is inheritably *simple* as a 0-dimensional geometry object.

`MULTIPOINT`s are *simple* if no two coordinates (`POINT`s) are equal (have identical coordinate values).

A `LINESTRING` is *simple* if it does not pass through the same `POINT` twice (except for the endpoints, in which case it is referred to as a linear ring and additionally considered closed).



(a)

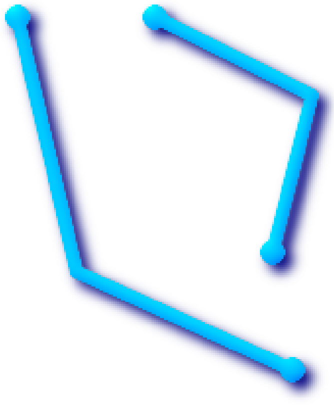
(b)

(c)

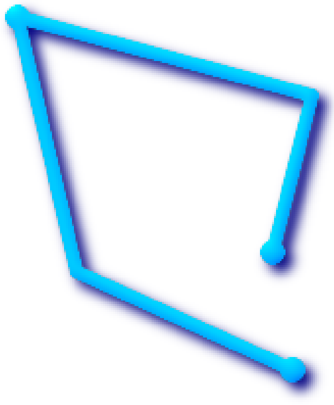
(d)

(a) and **(c)** are simple `LINESTRING`s, **(b)** and **(d)** are not.

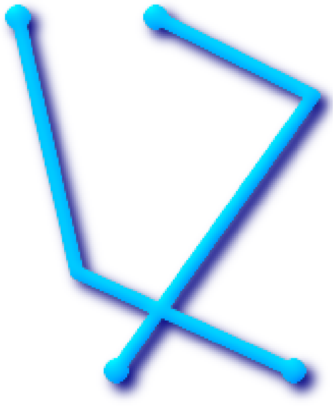
A `MULTILINESTRING` is *simple* only if all of its elements are simple and the only intersection between any two elements occurs at `POINT`s that are on the boundaries of both elements.



(e)



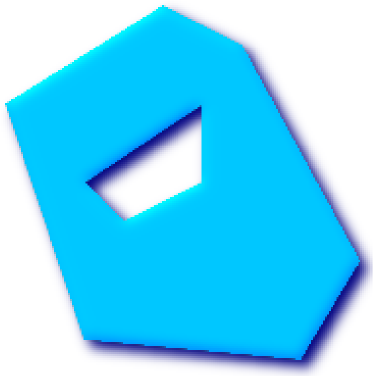
(f)



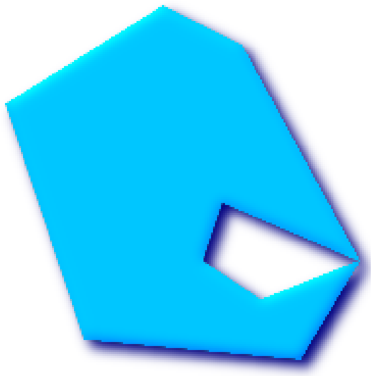
(g)

(e) and **(f)** are simple MULTILINESTRINGs, **(g)** is not.

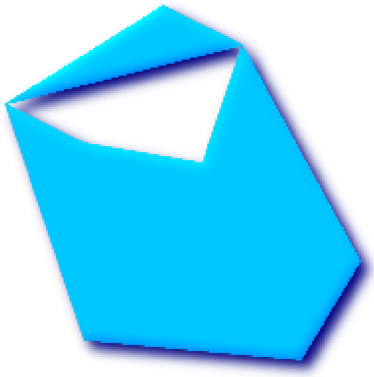
By definition, a POLYGON is always *simple*. It is *valid* if no two rings in the boundary (made up of an exterior ring and interior rings) cross. The boundary of a POLYGON may intersect at a POINT but only as a tangent (i.e. not on a line). A POLYGON may not have cut lines or spikes and the interior rings must be contained entirely within the exterior ring.



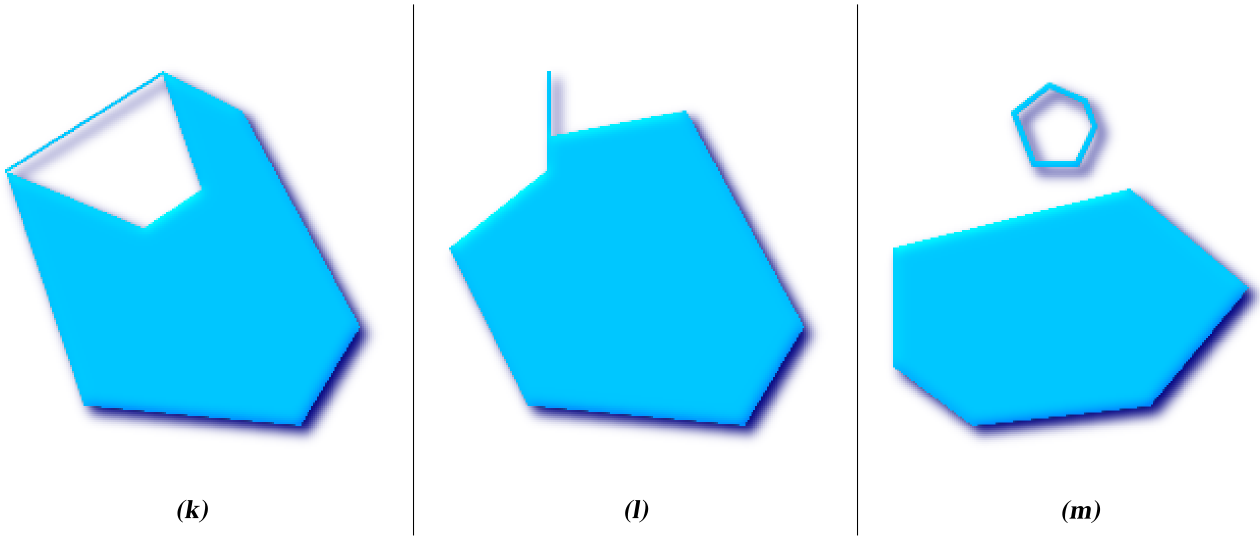
(h)



(i)



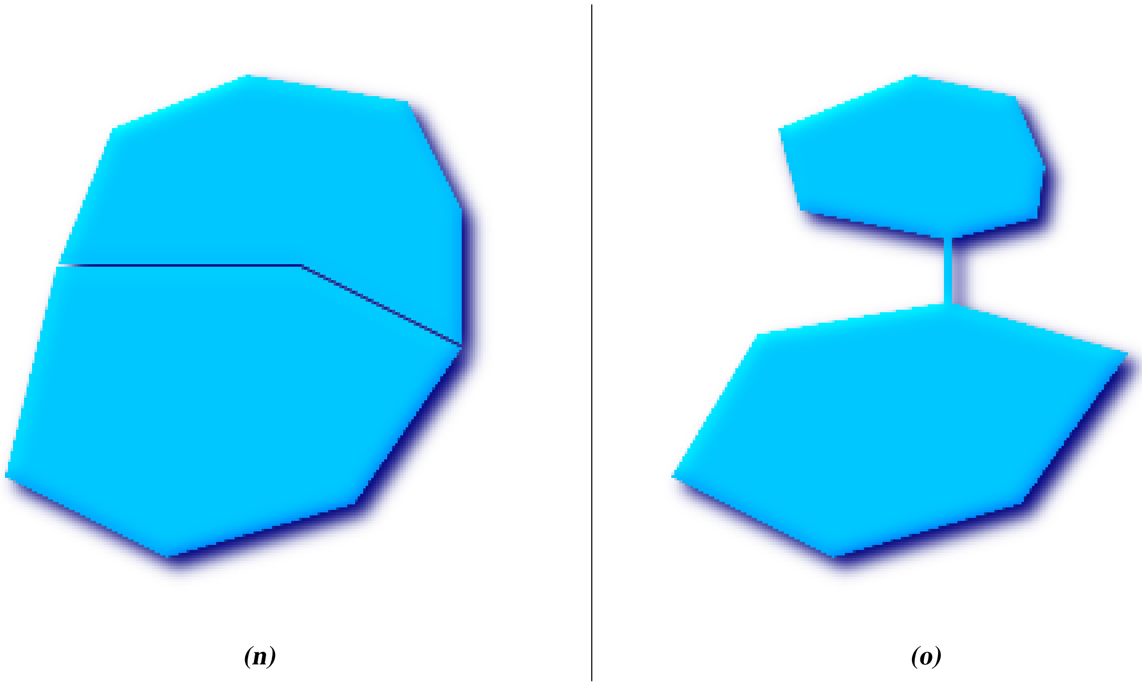
(j)



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

(h) and **(i)** are valid `POLYGON`s, **(j-m)** cannot be represented as single `POLYGON`s, but **(j)** and **(m)** could be represented as a valid `MULTIPOLYGON`.

A `MULTIPOLYGON` is *valid* if and only if all of its elements are valid and the interiors of no two elements intersect. The boundaries of any two elements may touch, but only at a finite number of `POINT`s.



(n) *(o)*

(n) and **(o)** are not valid `MULTIPOLYGON`s.

Most of the functions implemented by the GEOS library rely on the assumption that your geometries are valid as specified by the OpenGIS Simple Feature Specification. To check simplicity or validity of geometries you can use the ST_IsSimple() and ST_IsValid()

```
-- Typically, it doesn't make sense to check
-- for validity on linear features since it will always return TRUE.
-- But in this example, PostGIS extends the definition of the OGC IsValid
-- by returning false if a LinearRing (start and end points are the same)
-- has less than 2 vertices.
gisdb=# SELECT
   ST_IsValid('LINESTRING(0 0, 1 1)'),
   ST_IsValid('LINESTRING(0 0, 0 0)');

 st_isvalid | st_isvalid
------------+-----------
      t     |     f
```

By default, PostGIS does not apply this validity check on geometry input, because testing for validity needs lots of CPU time for complex geometries, especially polygons. If you do not trust your data sources, you can manually enforce such a check to your tables by adding a check constraint:

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

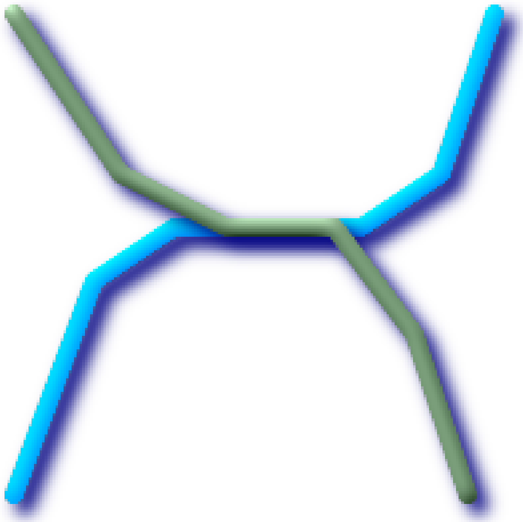
If you encounter any strange error messages such as "GEOS Intersection() threw an error!" or "JTS Intersection() threw an error!" when calling PostGIS functions with valid input geometries, you likely found an error in either PostGIS or one of the libraries it uses, and you should contact the PostGIS developers. The same is true if a PostGIS function returns an invalid geometry for valid input.

Note

Strictly compliant OGC geometries cannot have Z or M values. The ST_IsValid() function won't consider higher dimensioned geometries invalid! Invocations of AddGeometryColumn() will add a constraint checking geometry dimensions, so it is enough to specify 2 there.

4.3.6 Dimensionally Extended 9 Intersection Model (DE-9IM)

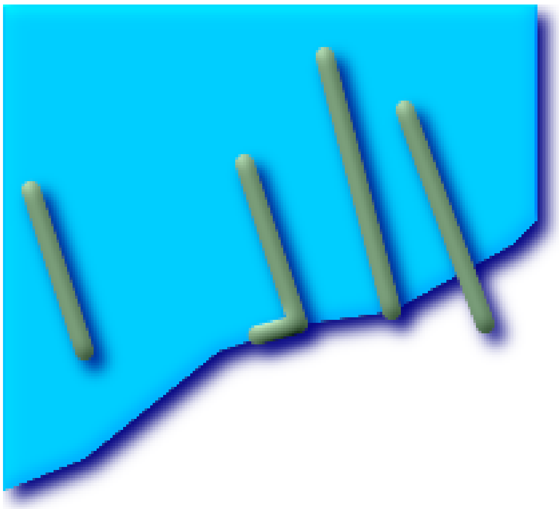
It is sometimes the case that the typical spatial predicates (ST_Contains, ST_Crosses, ST_Intersects, ST_Touches, ...) are insufficient in and of themselves to adequately provide that desired spatial filter.



For example, consider a linear dataset representing a road network. It may be the task of a GIS analyst to identify all road segments that cross each other, not at a point, but on a line, perhaps invalidating some business rule. In this case, ST_Crosses does not adequately provide the necessary spatial filter since, for linear features, it returns `true` only where they cross at a point.

One two-step solution might be to first perform the actual intersection (ST_Intersection) of pairs of road segments that spatially intersect (ST_Intersects), and then compare the intersection's ST_GeometryType with '`LINESTRING`' (properly dealing with cases that return `GEOMETRYCOLLECTION`s of `[MULTI]POINT`s, `[MULTI]LINESTRING`s, etc.).

A more elegant / faster solution may indeed be desirable.



A second [theoretical] example may be that of a GIS analyst trying to locate all wharfs or docks that intersect a lake's boundary on a line and where only one end of the wharf is up on shore. In other words, where a wharf is within, but not completely within a lake, intersecting the boundary of a lake on a line, and where the wharf's endpoints are both completely within and on the boundary of the lake. The analyst may need to use a combination of spatial predicates to isolate the sought after 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

  ... (needless to say, this could get quite complicated)

So enters the Dimensionally Extended 9 Intersection Model, or DE-9IM for short.

4.3.6.1 Theory

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."

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` are the two endpoints. For `POLYGON`s, the boundary is the linework that make up the exterior and interior rings.

Interior

The interior of a geometry are those points of a geometry that are left when the boundary is removed. For `POINT`s, the interior is the `POINT` itself. The interior of a `LINESTRING` are the set of real points between the endpoints. For `POLYGON`s, the interior is the areal surface inside the polygon.

Exterior

The exterior of a geometry is the universe, an areal surface, not on the interior or boundary of the geometry.

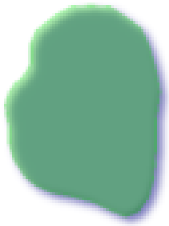
Given geometry *a*, where the *I(a)*, *B(a)*, and *E(a)* are the *Interior*, *Boundary*, and *Exterior* of a, the mathematical representation of the matrix is:

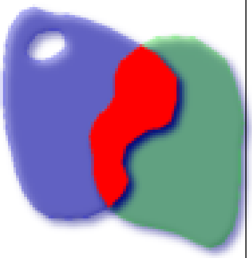
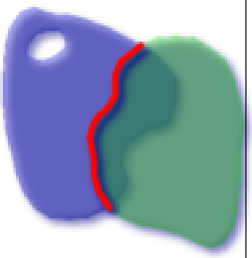
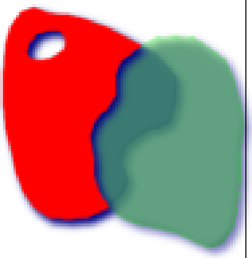
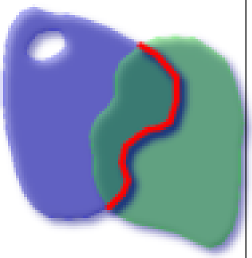
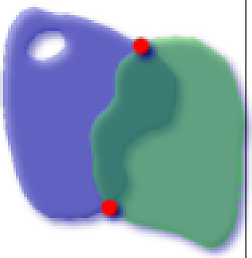
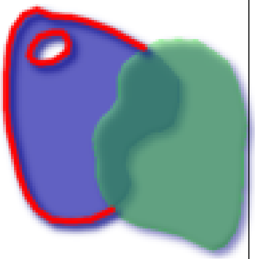
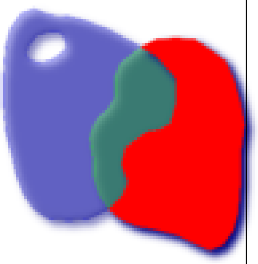
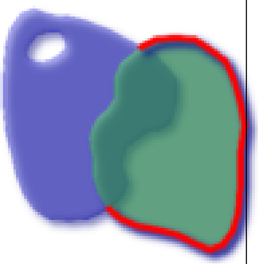
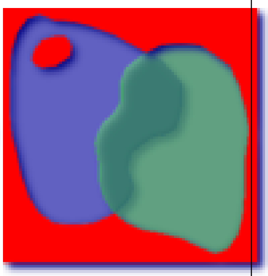
	Interior	**Boundary**	**Exterior**
Interior	*dim(I(a)∩I(b))*	*dim(I(a)∩B(b))*	*dim(I(a)∩E(b))*
Boundary	*dim(B(a)∩I(b))*	*dim(B(a)∩B(b))*	*dim(B(a)∩E(b))*
Exterior	*dim(E(a)∩I(b))*	*dim(E(a)∩B(b))*	*dim(E(a)∩E(b))*

Where *dim(a)* is the dimension of *a* as specified by ST_Dimension but has the domain of `{0,1,2,T,F,*}`

- `0` => point
- `1` => line
- `2` => area
- `T` => `{0,1,2}`
- `F` => empty set
- `*` => don't care

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

	Interior	Boundary	Exterior
Interior	 *dim(...)* = **2**	 *dim(...)* = **1**	 *dim(...)* = **2**
Boundary	 *dim(...)* = **1**	 *dim(...)* = **0**	 *dim(...)* = **1**
Exterior	 *dim(...)* = **2**	 *dim(...)* = **1**	 *dim(...)* = **2**

Read from left to right and from top to bottom, the dimensional matrix is represented, '**212101212**'.

A relate matrix that would therefore represent our first example of two lines that intersect on a line would be: '**1*1***1****'

```
-- Identify road segments that cross on 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**');
```

A relate matrix that represents the second example of wharfs partly on the lake's shoreline would be '**102101FF2**'

```
-- Identify wharfs 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');
```

For more information or reading, see:

- OpenGIS Simple Features Implementation Specification for SQL (version 1.1, section 2.1.13.2)
- Dimensionally Extended Nine-Intersection Model (DE-9IM) by Christian Strobl
- GeoTools: Dimensionally Extended Nine-Intersection Matrix
- *Encyclopedia of GIS* By Hui Xiong

4.4 Loading GIS Data

Once you have created a spatial table, you are ready to upload GIS data to the database. Currently, there are two ways to get data into a PostGIS/PostgreSQL database: using formatted SQL statements or using the Shape file loader/dumper.

4.4.1 Using SQL

If you can convert your data to a text representation, then using formatted SQL might be the easiest way to get your data into PostGIS. As with Oracle and other SQL databases, data can be bulk loaded by piping a large text file full of SQL "INSERT" statements into the SQL terminal monitor.

A data upload file (`roads.sql` for example) might look like this:

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

The data file can be piped into PostgreSQL very easily using the "psql" SQL terminal monitor:

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

4.4.2 Using the Loader

The `shp2pgsql` data loader converts ESRI Shape files into SQL suitable for insertion into a PostGIS/PostgreSQL database either in geometry or geography format. The loader has several operating modes distinguished by command line flags:

In addition to the shp2pgsql command-line loader, there is an `shp2pgsql-gui` graphical interface with most of the options as the command-line loader, but may be easier to use for one-off non-scripted loading or if you are new to PostGIS. It can also be configured as a plugin to PgAdminIII.

(c|a|d|p) These are mutually exclusive options:

-c Creates a new table and populates it from the shapefile. *This is the default mode.*

-a Appends data from the Shape file into the database table. Note that to use this option to load multiple files, the files must have the same attributes and same data types.

-d Drops the database table before creating a new table with the data in the Shape file.

-p Only produces the table creation SQL code, without adding any actual data. This can be used if you need to completely separate the table creation and data loading steps.

-? Display help screen.

-D Use the PostgreSQL "dump" format for the output data. This can be combined with -a, -c and -d. It is much faster to load than the default "insert" SQL format. Use this for very large data sets.

-s <SRID> Creates and populates the geometry tables with the specified SRID.

-k Keep identifiers' case (column, schema and attributes). Note that attributes in Shapefile are all UPPERCASE.

-i Coerce all integers to standard 32-bit integers, do not create 64-bit bigints, even if the DBF header signature appears to warrant it.

-I Create a GiST index on the geometry column.

-w Output WKT format, for use with older (0.x) versions of PostGIS. Note that this will introduce coordinate drifts and will drop M values from shapefiles.

-W <encoding> Specify encoding of the input data (dbf file). When used, all attributes of the dbf are converted from the specified encoding to UTF8. The resulting SQL output will contain a `SET CLIENT_ENCODING to UTF8` command, so that the backend will be able to reconvert from UTF8 to whatever encoding the database is configured to use internally.

-N <policy> NULL geometries handling policy (insert*,skip,abort)

-n -n Only import DBF file. If your data has no corresponding shapefile, it will automatically switch to this mode and load just the dbf. So setting this flag is only needed if you have a full shapefile set, and you only want the attribute data and no geometry.

-G Use geography type instead of geometry (requires lon/lat data) in WGS84 long lat (SRID=4326)

An example session using the loader to create an input file and uploading it might look like this:

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

A conversion and upload can be done all in one step using UNIX pipes:

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

4.5 Retrieving GIS Data

Data can be extracted from the database using either SQL or the Shape file loader/dumper. In the section on SQL we will discuss some of the operators available to do comparisons and queries on spatial tables.

4.5.1 Using SQL

The most straightforward means of pulling data out of the database is to use a SQL select query to reduce the number of RECORDS and COLUMNS returned and dump the resulting columns into a parsable text file:

```
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)
```

However, there will be times when some kind of restriction is necessary to cut down the number of fields returned. In the case of attribute-based restrictions, just use the same SQL syntax as normal with a non-spatial table. In the case of spatial restrictions, the following operators are available/useful:

&& This operator tells whether the bounding box of one geometry intersects the bounding box of another.

~= This operators tests whether two geometries are geometrically identical. For example, if 'POLYGON((0 0,1 1,1 0,0 0))' is the same as 'POLYGON((0 0,1 1,1 0,0 0))' (it is).

= This operator is a little more naive, it only tests whether the bounding boxes of two geometries are the same.

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 by using the "GeomFromText()" function. So, for example:

```
SELECT road_id, road_name
  FROM roads
  WHERE roads_geom ~= ST_GeomFromText('LINESTRING(191232 243118,191108 243242)');
```

The above query would return the single record from the "ROADS_GEOM" table in which the geometry was equal to that value.

When using the "&&" operator, you can specify either a BOX3D as the comparison feature or a GEOMETRY. When you specify a GEOMETRY, however, its bounding box will be used for the comparison.

```
SELECT road_id, road_name
FROM roads
WHERE roads_geom && ST_GeomFromText('POLYGON((...))');
```

The above query will use the bounding box of the polygon for comparison purposes.

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. 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 && SetSRID('BOX3D(191232 243117,191232 243119)'::box3d,-1);
```

Note the use of the SRID, to specify the projection of the BOX3D. The value -1 is used to indicate no specified SRID.

4.5.2 Using the Dumper

The `pgsql2shp` table dumper connects directly to the database and converts a table (possibly defined by a query) into a shape file. The basic syntax is:

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

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

The commandline options are:

-f <filename> Write the output to a particular filename.

-h <host> The database host to connect to.

-p <port> The port to connect to on the database host.

-P <password> The password to use when connecting to the database.

-u <user> The username to use when connecting to the database.

-g <geometry column> In the case of tables with multiple geometry columns, the geometry column to use when writing the shape file.

-b Use a binary cursor. This will make the operation faster, but will not work if any NON-geometry attribute in the table lacks a cast to text.

-r Raw mode. Do not drop the `gid` field, or escape column names.

-d For backward compatibility: write a 3-dimensional shape file when dumping from old (pre-1.0.0) postgis databases (the default is to write a 2-dimensional shape file in that case). Starting from postgis-1.0.0+, dimensions are fully encoded.

4.6 Building Indexes

Indexes are what make using a spatial database for large data sets possible. Without indexing, any search for a feature would require a "sequential scan" of every record in the database. Indexing speeds up searching by organizing the data into a search tree which can be quickly traversed to find a particular record. PostgreSQL supports three kinds of indexes by default: B-Tree indexes, R-Tree indexes, and GiST indexes.

- B-Trees are used for data which can be sorted along one axis; for example, numbers, letters, dates. GIS data cannot be rationally sorted along one axis (which is greater, (0,0) or (0,1) or (1,0)?) so B-Tree indexing is of no use for us.
- R-Trees break up data into rectangles, and sub-rectangles, and sub-sub rectangles, etc. R-Trees are used by some spatial databases to index GIS data, but the PostgreSQL R-Tree implementation is not as robust as the GiST implementation.
- GiST (Generalized Search Trees) indexes break up data into "things to one side", "things which overlap", "things which are inside" and can be used on a wide range of data-types, including GIS data. PostGIS uses an R-Tree index implemented on top of GiST to index GIS data.

4.6.1 GiST Indexes

GiST stands for "Generalized Search Tree" and is a generic form of indexing. In addition to GIS indexing, GiST is used to speed up searches on all kinds of irregular data structures (integer arrays, spectral data, etc) which are not amenable to normal B-Tree indexing.

Once a GIS data table exceeds a few thousand rows, you will want to build an index to speed up spatial searches of the data (unless all your searches are based on attributes, in which case you'll want to build a normal index on the attribute fields).

The syntax for building a GiST index on a "geometry" column is as follows:

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

Building a spatial index is a computationally intensive exercise: on tables of around 1 million rows, on a 300MHz Solaris machine, we have found building a GiST index takes about 1 hour. After building an index, it is important to force PostgreSQL to collect table statistics, which are used to optimize query plans:

```
VACUUM ANALYZE [table_name] [(column_name)];
-- This is only needed for PostgreSQL 7.4 installations and below
SELECT UPDATE_GEOMETRY_STATS([table_name], [column_name]);
```

GiST indexes have two advantages over R-Tree indexes in PostgreSQL. Firstly, GiST indexes are "null safe", meaning they can index columns which include null values. Secondly, GiST indexes support the concept of "lossiness" which is important when dealing with GIS objects larger than the PostgreSQL 8K page size. Lossiness allows PostgreSQL to store only the "important" part of an object in an index -- in the case of GIS objects, just the bounding box. GIS objects larger than 8K will cause R-Tree indexes to fail in the process of being built.

4.6.2 Using Indexes

Ordinarily, indexes invisibly speed up data access: once the index is built, the query planner transparently decides when to use index information to speed up a query plan. Unfortunately, the PostgreSQL query planner does not optimize the use of GiST indexes well, so sometimes searches which should use a spatial index instead default to a sequence scan of the whole table.

If you find your spatial indexes are not being used (or your attribute indexes, for that matter) there are a couple things you can do:

- Firstly, 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. For PostgreSQL 7.4 installations and below this is done by running **update_geometry_stats([table_name, column_name])** (compute distribution) and **VACUUM ANALYZE [table_name] [column_name]** (compute number of values). Starting with PostgreSQL 8.0 running **VACUUM ANALYZE** will do both operations. You should regularly vacuum your databases anyways -- many PostgreSQL DBAs have **VACUUM** run as an off-peak cron job on a regular basis.
- If vacuuming does not work, you can force the planner to use the index information by using the **SET ENABLE_SEQSCAN=OFF** command. You should only use this command sparingly, and only on spatially indexed queries: generally speaking, the planner knows better than you do about when to use normal B-Tree indexes. Once you have run your query, you should consider setting `ENABLE_SEQSCAN` back on, so that other queries will utilize the planner as normal.

Note
As of version 0.6, it should not be necessary to force the planner to use the index with `ENABLE_SEQSCAN`.

- If you find the planner wrong about the cost of sequential vs index scans try reducing the value of random_page_cost in postgresql.conf or using SET random_page_cost=#. Default value for the parameter is 4, try setting it to 1 or 2. Decrementing the value makes the planner more inclined of using Index scans.

4.7 Complex Queries

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

4.7.1 Taking Advantage of Indexes

When constructing a query it is important to remember that only the bounding-box-based operators such as && can take advantage of the GiST spatial index. Functions such as `distance()` cannot use the index to optimize their operation. For example, the following query would be quite slow on a large table:

```
SELECT the_geom
FROM geom_table
WHERE ST_Distance(the_geom, ST_GeomFromText('POINT(100000 200000)')) < 100
```

This query is selecting 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 our specified point, ie. one `ST_Distance()` calculation for each row in the table. We can avoid this by using the && operator to reduce the number of distance calculations required:

```
SELECT the_geom
FROM geom_table
WHERE the_geom && 'BOX3D(90900 190900, 100100 200100)'::box3d
  AND
ST_Distance(the_geom, ST_GeomFromText('POINT(100000 200000)')) < 100
```

This query selects the same geometries, but it does it in a more efficient way. Assuming there is a GiST index on the_geom, the query planner will recognize that it can use the index to reduce the number of rows before calculating the result of the `distance()` function. Notice that the `BOX3D` geometry which is used in the && operation is a 200 unit square box centered on the original point - this is our "query box". The && operator uses the index to quickly reduce the result set down to only those geometries which have bounding boxes that overlap the "query box". Assuming that our query box is much smaller than the extents of the entire geometry table, this will drastically reduce the number of distance calculations that need to be done.

Change in Behavior
As of PostGIS 1.3.0, most of the Geometry Relationship Functions, with the notable exceptions of ST_Disjoint and ST_Relate, include implicit bounding box overlap operators.

4.7.2 Examples of Spatial SQL

The examples in this section will make use of two tables, a table of linear roads, and a table of polygonal municipality boundaries. The table definitions for the `bc_roads` table is:

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

The table definition for the `bc_municipality` table is:

```
Column     | Type              | Description
-----------+-------------------+-------------------
gid        | integer           | Unique ID
code       | integer           | Unique ID
name       | character varying | City / Town Name
the_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(the_geom))/1000 AS km_roads FROM bc_roads;

   km_roads
   ------------------
   70842.1243039643
   (1 row)
   ```

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 area):

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

   hectares
   ------------------
   32657.9103824927
   (1 row)
   ```

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

   This query brings a spatial measurement into the query condition. There are several ways of approaching this problem, but the most efficient is below:

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

name           | hectares
---------------+-----------------
TUMBLER RIDGE  | 155020.02556131
(1 row)
```

   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 we could separately index for performance. By ordering the results in a descending direction, and them using the PostgreSQL "LIMIT" command we can easily pick off 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", because we are bringing together data from two tables (doing a join) but using a spatial interaction condition ("contained") as the join condition rather than the usual relational approach of joining on a common key:

```
SELECT
  m.name,
  sum(ST_Length(r.the_geom))/1000 as roads_km
FROM
  bc_roads AS r,
  bc_municipality AS m
WHERE
  ST_Contains(m.the_geom,r.the_geom)
GROUP BY m.name
ORDER BY roads_km;

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

   This query takes a while, because every road in the table is summarized into the final result (about 250K roads for our particular example table). For smaller overlays (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 actually 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.the_geom, m.the_geom) AS intersection_geom,
  ST_Length(r.the_geom) AS rd_orig_length,
  r.*
```

```
FROM
  bc_roads AS r,
  bc_municipality AS m
WHERE  m.name = 'PRINCE GEORGE' AND ST_Intersects(r.the_geom, m.the_geom);
```

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

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

kilometers
------------------
4.89151904172838
(1 row)
```

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

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

gid  | name         | area
-----+--------------+------------------
12   | SPALLUMCHEEN | 257374619.430216
(1 row)
```

Chapter 5

Using PostGIS: Building Applications

5.1 Using MapServer

The Minnesota MapServer is an internet web-mapping server which conforms to the OpenGIS Web Mapping Server specification.

- The MapServer homepage is at http://mapserver.org.
- The OpenGIS Web Map Specification is at http://www.opengeospatial.org/standards/wms.

5.1.1 Basic Usage

To use PostGIS with MapServer, you will need to know about how to configure MapServer, which is beyond the scope of this documentation. This section will cover specific PostGIS issues and configuration details.

To use PostGIS with MapServer, you will need:

- Version 0.6 or newer of PostGIS.
- Version 3.5 or newer of MapServer.

MapServer accesses PostGIS/PostgreSQL data like any other PostgreSQL client -- using the `libpq` interface. This means that MapServer can be installed on any machine with network access to the PostGIS server, and use PostGIS as a source of data. The faster the connection between the systems, the better.

1. Compile and install MapServer, with whatever options you desire, including the "--with-postgis" configuration option.
2. In your MapServer map file, add a PostGIS layer. For example:

```
LAYER
  CONNECTIONTYPE postgis
  NAME "widehighways"
  # Connect to a remote spatial database
  CONNECTION "user=dbuser dbname=gisdatabase host=bigserver"
  PROCESSING "CLOSE_CONNECTION=DEFER"
  # Get the lines from the ’geom’ column of the ’roads’ table
  DATA "geom from roads using srid=4326 using unique gid"
  STATUS ON
  TYPE LINE
  # Of the lines in the extents, only render the wide highways
  FILTER "type = ’highway’ and numlanes >= 4"
  CLASS
    # Make the superhighways brighter and 2 pixels wide
    EXPRESSION ([numlanes] >= 6)
    STYLE
      COLOR 255 22 22
      WIDTH 2
    END
  END
  CLASS
    # All the rest are darker and only 1 pixel wide
    EXPRESSION ([numlanes] < 6)
    STYLE
      COLOR 205 92 82
    END
  END
END
```

In the example above, the PostGIS-specific directives are as follows:

CONNECTIONTYPE For PostGIS layers, this is always "postgis".

CONNECTION The database connection is governed by the a ’connection string’ which is a standard set of keys and values like this (with the default values in <>):

user=<username> password=<password> dbname=<username> hostname=<server> port=<5432>

An empty connection string is still valid, and any of the key/value pairs can be omitted. At a minimum you will generally supply the database name and username to connect with.

DATA The form of this parameter is "<geocolumn> from <tablename> using srid=<srid> using unique <primary key>" where the column is the spatial column to be rendered to the map, the SRID is SRID used by the column and the primary key is the table primary key (or any other uniquely-valued column with an index).

You can omit the "using srid" and "using unique" clauses and MapServer will automatically determine the correct values if possible, but at the cost of running a few extra queries on the server for each map draw.

PROCESSING Putting in a CLOSE_CONNECTION=DEFER if you have multiple layers reuses existing connections instead of closing them. This improves speed. Refer to for MapServer PostGIS Performance Tips for a more detailed explanation.

FILTER The filter must be a valid SQL string corresponding to the logic normally following the "WHERE" keyword in a SQL query. So, for example, to render only roads with 6 or more lanes, use a filter of "num_lanes >= 6".

3. In your spatial database, ensure you have spatial (GiST) indexes built for any the layers you will be drawing.

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

4. If you will be querying your layers using MapServer you will also need to use the "using unique" clause in your DATA statement.

   MapServer requires unique identifiers for each spatial record when doing queries, and the PostGIS module of MapServer uses the unique value you specify in order to provide these unique identifiers. Using the table primary key is the best practice.

5.1.2 Frequently Asked Questions

1. *When I use an `EXPRESSION` in my map file, the condition never returns as true, even though I know the values exist in my table.*

   Unlike shape files, PostGIS field names have to be referenced in EXPRESSIONS using *lower case*.

```
EXPRESSION ([numlanes] >= 6)
```

2. *The FILTER I use for my Shape files is not working for my PostGIS table of the same data.*

   Unlike shape files, filters for PostGIS layers use SQL syntax (they are appended to the SQL statement the PostGIS connector generates for drawing layers in MapServer).

```
FILTER "type = 'highway' and numlanes >= 4"
```

3. *My PostGIS layer draws much slower than my Shape file layer, is this normal?*

   In general, the more features you are drawing into a given map, the more likely it is that PostGIS will be slower than Shape files. For maps with relatively few features (100s), PostGIS will often be faster. For maps with high feature density (1000s), PostGIS will always be slower. If you are finding substantial draw performance problems, it is possible that you have not built a spatial index on your table.

   ```
   postgis# CREATE INDEX geotable_gix ON geotable USING GIST ( geocolumn );
   postgis# VACUUM ANALYZE;
   ```

4. *My PostGIS layer draws fine, but queries are really slow. What is wrong?*

   For queries to be fast, you must have a unique key for your spatial table and you must have an index on that unique key.You can specify what unique key for mapserver to use with the `USING UNIQUE` clause in your `DATA` line:

   ```
   DATA "the_geom FROM geotable USING UNIQUE gid"
   ```

5. *Can I use "geography" columns (new in PostGIS 1.5) as a source for MapServer layers?*

   Yes! MapServer understands geography columns as being the same as geometry columns, but always using an SRID of 4326. Just make sure to include a "using srid=4326" clause in your `DATA` statement. Everything else works exactly the same as with geometry.

   ```
   DATA "the_geog FROM geogtable USING SRID=4326 USING UNIQUE gid"
   ```

5.1.3 Advanced Usage

The `USING` pseudo-SQL clause is used to add some information to help mapserver understand the results of more complex queries. More specifically, when either a view or a subselect is used as the source table (the thing to the right of "FROM" in a `DATA` definition) it is more difficult for mapserver to automatically determine a unique identifier for each row and also the SRID for the table. The `USING` clause can provide mapserver with these two pieces of information as follows:

```
DATA "the_geom FROM (
  SELECT
    table1.the_geom AS the_geom,
    table1.gid AS gid,
    table2.data AS data
  FROM table1
  LEFT JOIN table2
  ON table1.id = table2.id
) AS new_table USING UNIQUE gid USING SRID=-1"
```

USING UNIQUE <uniqueid> MapServer requires a unique id for each row in order to identify the row when doing map queries. Normally it identifies the primary key from the system tables. However, views and subselects don't automatically have an known unique column. If you want to use MapServer's query functionality, you need to ensure your view or subselect includes a uniquely valued column, and declare it with `USING UNIQUE`. For example, you could explicitly select nee of the table's primary key values for this purpose, or any other column which is guaranteed to be unique for the result set.

> **Note**
> "Querying a Map" is the action of clicking on a map to ask for information about the map features in that location. Don't confuse "map queries" with the SQL query in a `DATA` definition.

USING SRID=<srid> PostGIS needs to know which spatial referencing system is being used by the geometries in order to return the correct data back to MapServer. Normally it is possible to find this information in the "geometry_columns" table in the PostGIS database, however, this is not possible for tables which are created on the fly such as subselects and views. So the `USING SRID=` option allows the correct SRID to be specified in the `DATA` definition.

5.1.4 Examples

Lets start with a simple example and work our way up. Consider the following MapServer layer definition:

```
LAYER
  CONNECTIONTYPE postgis
  NAME "roads"
  CONNECTION "user=theuser password=thepass dbname=thedb host=theserver"
  DATA "the_geom from roads"
  STATUS ON
  TYPE LINE
  CLASS
    STYLE
      COLOR 0 0 0
    END
  END
END
```

This layer will display all the road geometries in the roads table as black lines.

Now lets say we want to show only the highways until we get zoomed in to at least a 1:100000 scale - the next two layers will achieve this effect:

```
LAYER
  CONNECTIONTYPE postgis
  CONNECTION "user=theuser password=thepass dbname=thedb host=theserver"
  PROCESSING "CLOSE_CONNECTION=DEFER"
  DATA "the_geom from roads"
  MINSCALE 100000
  STATUS ON
  TYPE LINE
  FILTER "road_type = ’highway’"
  CLASS
    COLOR 0 0 0
  END
END
LAYER
  CONNECTIONTYPE postgis
  CONNECTION "user=theuser password=thepass dbname=thedb host=theserver"
  PROCESSING "CLOSE_CONNECTION=DEFER"
  DATA "the_geom from roads"
  MAXSCALE 100000
  STATUS ON
  TYPE LINE
  CLASSITEM road_type
  CLASS
    EXPRESSION "highway"
    STYLE
      WIDTH 2
      COLOR 255 0 0
    END
  END
  CLASS
    STYLE
      COLOR 0 0 0
    END
  END
END
```

The first layer is used when the scale is greater than 1:100000, and displays only the roads of type "highway" as black lines. The `FILTER` option causes only roads of type "highway" to be displayed.

The second layer is used when the scale is less than 1:100000, and will display highways as double-thick red lines, and other roads as regular black lines.

So, we have done a couple of interesting things using only MapServer functionality, but our `DATA` SQL statement has remained simple. Suppose that the name of the road is stored in another table (for whatever reason) and we need to do a join to get it and label our roads.

```
LAYER
  CONNECTIONTYPE postgis
  CONNECTION "user=theuser password=thepass dbname=thedb host=theserver"
  DATA "the_geom FROM (SELECT roads.oid AS oid, roads.the_geom AS the_geom,
        road_names.name as name FROM roads LEFT JOIN road_names ON
        roads.road_name_id = road_names.road_name_id)
        AS named_roads USING UNIQUE oid USING SRID=-1"
  MAXSCALE 20000
  STATUS ON
  TYPE ANNOTATION
  LABELITEM name
  CLASS
    LABEL
      ANGLE auto
      SIZE 8
      COLOR 0 192 0
      TYPE truetype
      FONT arial
    END
  END
END
```

This annotation layer adds green labels to all the roads when the scale gets down to 1:20000 or less. It also demonstrates how to use an SQL join in a `DATA` definition.

5.2 Java Clients (JDBC)

Java clients can access PostGIS "geometry" objects in the PostgreSQL database either directly as text representations or using the JDBC extension objects bundled with PostGIS. In order to use the extension objects, the "postgis.jar" file must be in your CLASSPATH along with the "postgresql.jar" JDBC driver package.

```
import java.sql.*;
import java.util.*;
import java.lang.*;
import org.postgis.*;

public class JavaGIS {

public static void main(String[] args) {

  java.sql.Connection conn;

  try {
    /*
    * Load the JDBC driver and establish a connection.
    */
    Class.forName("org.postgresql.Driver");
    String url = "jdbc:postgresql://localhost:5432/database";
    conn = DriverManager.getConnection(url, "postgres", "");
    /*
    * Add the geometry types to the connection. Note that you
    * must cast the connection to the pgsql-specific connection
    * implementation before calling the addDataType() method.
    */
    ((org.postgresql.PGConnection)conn).addDataType("geometry",Class.forName("org.postgis. ↩
        PGgeometry"));
    ((org.postgresql.PGConnection)conn).addDataType("box3d",Class.forName("org.postgis. ↩
        PGbox3d"));

    /*
    * Create a statement and execute a select query.
    */
    Statement s = conn.createStatement();
    ResultSet r = s.executeQuery("select geom,id from geomtable");
    while( r.next() ) {
      /*
      * Retrieve the geometry as an object then cast it to the geometry type.
      * Print things out.
      */
      PGgeometry geom = (PGgeometry)r.getObject(1);
      int id = r.getInt(2);
      System.out.println("Row " + id + ":");
      System.out.println(geom.toString());
    }
    s.close();
    conn.close();
  }
catch( Exception e ) {
  e.printStackTrace();
  }
}
}
```

The "PGgeometry" object is a wrapper object which contains a specific topological geometry object (subclasses of the abstract class "Geometry") depending on the type: Point, LineString, Polygon, MultiPoint, MultiLineString, MultiPolygon.

```
PGgeometry geom = (PGgeometry)r.getObject(1);
if( geom.getType() == Geometry.POLYGON ) {
  Polygon pl = (Polygon)geom.getGeometry();
  for( int r = 0; r < pl.numRings(); r++) {
    LinearRing rng = pl.getRing(r);
    System.out.println("Ring: " + r);
    for( int p = 0; p < rng.numPoints(); p++ ) {
      Point pt = rng.getPoint(p);
      System.out.println("Point: " + p);
      System.out.println(pt.toString());
    }
  }
}
```

The JavaDoc for the extension objects provides a reference for the various data accessor functions in the geometric objects.

5.3 C Clients (libpq)

...

5.3.1 Text Cursors

...

5.3.2 Binary Cursors

...

Chapter 6

Performance tips

6.1 Small tables of large geometries

6.1.1 Problem description

Current PostgreSQL versions (including 8.0) suffer from a query optimizer weakness regarding TOAST tables. TOAST tables are a kind of "extension room" used to store large (in the sense of data size) values that do not fit into normal data pages (like long texts, images or complex geometries with lots of vertices), see http://www.postgresql.org/docs/current/interactive/storage-toast.html for more information).

The problem appears if you happen to have a table with rather large geometries, but not too much rows of them (like a table containing the boundaries of all European countries in high resolution). Then the table itself is small, but it uses lots of TOAST space. In our example case, the table itself had about 80 rows and used only 3 data pages, but the TOAST table used 8225 pages.

Now issue a query where you use the geometry operator && to search for a bounding box that matches only very few of those rows. Now the query optimizer sees that the table has only 3 pages and 80 rows. He estimates that a sequential scan on such a small table is much faster than using an index. And so he decides to ignore the GIST index. Usually, this estimation is correct. But in our case, the && operator has to fetch every geometry from disk to compare the bounding boxes, thus reading all TOAST pages, too.

To see whether your suffer from this bug, use the "EXPLAIN ANALYZE" postgresql command. For more information and the technical details, you can read the thread on the postgres performance mailing list: http://archives.postgresql.org/pgsql-performance/2005-02/msg00030.php

6.1.2 Workarounds

The PostgreSQL people are trying to solve this issue by making the query estimation TOAST-aware. For now, here are two workarounds:

The first workaround is to force the query planner to use the index. Send "SET enable_seqscan TO off;" to the server before issuing the query. This basically forces the query planner to avoid sequential scans whenever possible. So it uses the GIST index as usual. But this flag has to be set on every connection, and it causes the query planner to make misestimations in other cases, so you should "SET enable_seqscan TO on;" after the query.

The second workaround is to make the sequential scan as fast as the query planner thinks. This can be achieved by creating an additional column that "caches" the bbox, and matching against this. In our example, the commands are like:

```
SELECT AddGeometryColumn('myschema','mytable','bbox','4326','GEOMETRY','2');
UPDATE mytable SET bbox = ST_Envelope(ST_Force_2d(the_geom));
```

Now change your query to use the && operator against bbox instead of geom_column, like:

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

Of course, if you change or add rows to mytable, you have to keep the bbox "in sync". The most transparent way to do this would be triggers, but you also can modify your application to keep the bbox column current or run the UPDATE query above after every modification.

6.2 CLUSTERing on geometry indices

For tables that are mostly read-only, and where a single index is used for the majority of queries, PostgreSQL offers the CLUSTER command. This command physically reorders all the data rows in the same order as the index criteria, yielding two performance advantages: First, for index range scans, the number of seeks on the data table is drastically reduced. Second, if your working set concentrates to some small intervals on the indices, you have a more efficient caching because the data rows are spread along fewer data pages. (Feel invited to read the CLUSTER command documentation from the PostgreSQL manual at this point.)

However, currently PostgreSQL does not allow clustering on PostGIS GIST indices because GIST indices simply ignores NULL values, you get an error message like:

```
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 "the_geom" NOT NULL.
```

As the HINT message tells you, one can work around this deficiency by adding a "not null" constraint to the table:

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

Of course, this will not work if you in fact need NULL values in your geometry column. Additionally, you must use the above method to add the constraint, using a CHECK constraint like "ALTER TABLE blubb ADD CHECK (geometry is not null);" will not work.

6.3 Avoiding dimension conversion

Sometimes, you happen to have 3D or 4D data in your table, but always access it using OpenGIS compliant ST_AsText() or ST_AsBinary() functions that only output 2D geometries. They do this by internally calling the ST_Force_2d() function, which introduces a significant overhead for large geometries. To avoid this overhead, it may be feasible to pre-drop those additional dimensions once and forever:

```
UPDATE mytable SET the_geom = ST_Force_2d(the_geom);
VACUUM FULL ANALYZE mytable;
```

Note that if you added your geometry column using AddGeometryColumn() there'll be a constraint on geometry dimension. To bypass it you will need to drop the constraint. Remember to update the entry in the geometry_columns table and recreate the constraint afterwards.

In case of large tables, it may be wise to divide this UPDATE into smaller portions by constraining the UPDATE to a part of the table via a WHERE clause and your primary key or another feasible criteria, and running a simple "VACUUM;" between your UPDATEs. This drastically reduces the need for temporary disk space. Additionally, if you have mixed dimension geometries, restricting the UPDATE by "WHERE dimension(the_geom)>2" skips re-writing of geometries that already are in 2D.

6.4 Tuning your configuration

These tips are taken from Kevin Neufeld's presentation "Tips for the PostGIS Power User" at the FOSS4G 2007 conference. Depending on your use of PostGIS (for example, static data and complex analysis vs frequently updated data and lots of users) these changes can provide significant speedups to your queries.

For a more tips (and better formatting), the original presentation is at http://2007.foss4g.org/presentations/view.php?abstract_id=117.

6.4.1 Startup

These settings are configured in postgresql.conf:

checkpoint_segment_size (this setting is obsolete in newer versions of PostgreSQL) got replaced with many configurations with names starting with checkpoint and WAL.

- # of WAL files = 16MB each; default is 3
- Set to at least 10 or 30 for databases with heavy write activity, or more for large database loads. Another article on the topic worth reading Greg Smith: Checkpoint and Background writer
- Possibly store the xlog on a separate disk device

constraint_exclusion

- Default: off (prior to PostgreSQL 8.4 and for PostgreSQL 8.4+ is set to partition)
- This is generally used for table partitioning. If you are running PostgreSQL versions below 8.4, set to "on" to ensure the query planner will optimize as desired. As of PostgreSQL 8.4, 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: ~32MB
- Set to about 1/3 to 3/4 of available RAM

6.4.2 Runtime

work_mem (the memory used for sort operations and complex queries)

- Default: 1MB
- 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 1200000;
```

maintenance_work_mem (used for VACUUM, CREATE INDEX, etc.)

- Default: 16MB
- Generally too low - ties up I/O, locks objects while swapping memory
- Recommend 32MB to 256MB on production servers w/lots of RAM, but depends on the # of concurrent users. If you have lots of RAM and few developers:

```
SET maintainence_work_mem TO 1200000;
```

Chapter 7

PostGIS Reference

The functions given below are the ones which a user of PostGIS is likely to need. There are other functions which are required support functions to the PostGIS objects which are not of use to a general user.

Note

PostGIS has begun a transition from the existing naming convention to an SQL-MM-centric convention. As a result, most of the functions that you know and love have been renamed using the standard spatial type (ST) prefix. Previous functions are still available, though are not listed in this document where updated functions are equivalent. The non ST_ functions not listed in this documentation are deprecated and will be removed in a future release so STOP USING THEM.

7.1 PostgreSQL PostGIS Types

7.1.1 box2d

box2d — A box composed of x min, ymin, xmax, ymax. Often used to return the 2d enclosing box of a geometry.

Description

box2d is a spatial data type used to represent the enclosing box of a geometry or set of geometries. ST_Extent in earlier versions prior to PostGIS 1.4 would return a box2d.

7.1.2 box3d

box3d — A box composed of x min, ymin, zmin, xmax, ymax, zmax. Often used to return the 3d extent of a geometry or collection of geometries.

Description

box3d is a postgis spatial data type used to represent the enclosing box of a geometry or set of geometries. ST_Extent3D returns a box3d object.

Casting Behavor

This section lists the automatic as well as explicit casts allowed for this data type

Cast To	Behavior
box	automatic
box2d	automatic
geometry	automatic

7.1.3 box3d_extent

box3d_extent — A box composed of x min, ymin, zmin, xmax, ymax, zmax. Often used to return the extent of a geometry.

Description

box3d_extent is a data type returned by ST_Extent. In versions prior to PostGIS 1.4, ST_Extent would return a box2d.

Casting Behavor

This section lists the automatic as well as explicit casts allowed for this data type

Cast To	Behavior
box2d	automatic
box3d	automatic
geometry	automatic

See Also

Section 8.5

7.1.4 geometry

geometry — Planar spatial data type.

Description

geometry is a fundamental postgis spatial data type used to represent a feature in the Euclidean coordinate system.

Casting Behavor

This section lists the automatic as well as explicit casts allowed for this data type

Cast To	Behavior
box	automatic
box2d	automatic
box3d	automatic
bytea	automatic
geography	automatic
text	automatic

See Also

Section 4.1

7.1.5 geometry_dump

geometry_dump — A spatial datatype with two fields - geom (holding a geometry object) and path[] (a 1-d array holding the position of the geometry within the dumped object.)

Description

geometry_dump is a compound data type consisting of a geometry object referenced by the .geom field and path[] a 1-dimensional integer array (starting at 1 e.g. path[1] to get first element) array that defines the navigation path within the dumped geometry to find this element. It is used by the ST_Dump* family of functions as an output type to explode a more complex geometry into its constituent parts and location of parts.

See Also

Section 8.4

7.1.6 geography

geography — Ellipsoidal spatial data type.

Description

geography is a spatial data type used to represent a feature in the round-earth coordinate system.

Casting Behavor

This section lists the automatic as well as explicit casts allowed for this data type

Cast To	Behavior
geometry	explicit

See Also

Section 8.3,Section 4.2

7.2 Management Functions

7.2.1 AddGeometryColumn

AddGeometryColumn — Adds a geometry column to an existing table of attributes.

Synopsis

text **AddGeometryColumn**(varchar table_name, varchar column_name, integer srid, varchar type, integer dimension);
text **AddGeometryColumn**(varchar schema_name, varchar table_name, varchar column_name, integer srid, varchar type, integer dimension);
text **AddGeometryColumn**(varchar catalog_name, varchar schema_name, varchar table_name, varchar column_name, integer srid, varchar type, integer dimension);

Description

Adds a geometry column to an existing table of attributes. The `schema_name` is the name of the table schema (unused for pre-schema PostgreSQL installations). The `srid` must be an integer value reference to an entry in the SPATIAL_REF_SYS table. The `type` must be an uppercase string corresponding to the geometry type, eg, 'POLYGON' or 'MULTILINESTRING'. An error is thrown if the schemaname doesn't exist (or not visible in the current search_path) or the specified SRID, geometry type, or dimension is invalid.

Note

Views and derivatively created spatial tables will need to be registered in geometry_columns manually, since AddGeometryColumn also adds a spatial column which is not needed when you already have a spatial column. Refer to Section 4.3.4.

✓ This method implements the OpenGIS 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

Examples

```
-- Create a new simple PostgreSQL table
postgis=# 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
postgis=# SELECT AddGeometryColumn ('my_schema','my_spatial_table','the_geom',4326,'POINT ↩
    ',2);

--Add a curvepolygon
SELECT AddGeometryColumn ('my_schema','my_spatial_table','the_geomcp',4326,'CURVEPOLYGON ↩
    ',2);

-- Describe the table again reveals the addition of a new "the_geom" column.
postgis=# \d my_schema.my_spatial_table
   Column    |   Type   |                                Modifiers

------------+----------+------------------------------------------------------------------------- ↩

 id          | integer  | not null default nextval('my_schema.my_spatial_table_id_seq':: ↩
     regclass)
 the_geom    | geometry |
 the_geomcp | geometry |
Check constraints:
  "enforce_dims_the_geom" CHECK (ndims(the_geom) = 2)
  "enforce_dims_the_geomcp" CHECK (ndims(the_geomcp) = 2)
  "enforce_geotype_the_geom" CHECK (geometrytype(the_geom) = 'POINT'::text OR
the_geom IS NULL)
  "enforce_geotype_the_geomcp" CHECK (geometrytype(the_geomcp) = 'CURVEPOLYGON
'::text OR the_geomcp IS NULL)
  "enforce_srid_the_geom" CHECK (srid(the_geom) = 4326)
  "enforce_srid_the_geomcp" CHECK (srid(the_geomcp) = 4326)
```

See Also

DropGeometryColumn, DropGeometryTable, Section 4.3.4

7.2.2 DropGeometryColumn

DropGeometryColumn — Removes a geometry column from a 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);

Description

Removes a geometry column from a spatial table. Note that schema_name will need to match the f_table_schema field of the table's row in the geometry_columns table.

✓ This method implements the OpenGIS 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

Examples

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

See Also

AddGeometryColumn, DropGeometryTable

7.2.3 DropGeometryTable

DropGeometryTable — Drops a table and all its references in 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);

Description

Drops a table and all its references in geometry_columns. Note: uses current_schema() on schema-aware pgsql installations if schema is not provided.

Examples

```
      SELECT DropGeometryTable ('my_schema','my_spatial_table');
      ----RESULT output ---
      my_schema.my_spatial_table dropped.
```

See Also

AddGeometryColumn, DropGeometryColumn

7.2.4 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.

Examples

```
SELECT PostGIS_Full_Version();
                 postgis_full_version
----------------------------------------------------------------------------------
 POSTGIS="1.3.3" GEOS="3.1.0-CAPI-1.5.0" PROJ="Rel. 4.4.9, 29 Oct 2004" USE_STATS
(1 row)
```

See Also

PostGIS_GEOS_Version, PostGIS_Lib_Version, PostGIS_LibXML_Version, PostGIS_PROJ_Version, PostGIS_Version

7.2.5 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
----------------------
 3.1.0-CAPI-1.5.0
(1 row)
```

See Also

PostGIS_Full_Version, PostGIS_Lib_Version, PostGIS_LibXML_Version, PostGIS_PROJ_Version, PostGIS_Version

7.2.6 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
----------------------
 2.7.6
(1 row)
```

See Also

PostGIS_Full_Version, PostGIS_Lib_Version, PostGIS_PROJ_Version, PostGIS_GEOS_Version, PostGIS_Version

7.2.7 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
------------------------
 2008-06-21 17:53:21
(1 row)
```

7.2.8 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
---------------------
 1.3.3
(1 row)
```

See Also

PostGIS_Full_Version, PostGIS_GEOS_Version, PostGIS_LibXML_Version, PostGIS_PROJ_Version, PostGIS_Version

7.2.9 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 PROJ4 library, or `NULL` if PROJ4 support is not enabled.

Examples

```
SELECT PostGIS_PROJ_Version();
  postgis_proj_version
-------------------------
 Rel. 4.4.9, 29 Oct 2004
(1 row)
```

See Also

PostGIS_Full_Version, PostGIS_GEOS_Version, PostGIS_Lib_Version, PostGIS_LibXML_Version, PostGIS_Version

7.2.10 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
-------------------------
 2007-08-18 09:09:26
(1 row)
```

See Also

PostGIS_Full_Version, PostGIS_GEOS_Version, PostGIS_Lib_Version, PostGIS_LibXML_Version, PostGIS_Version

7.2.11 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
-------------------------
 1.5.0SVN
(1 row)
```

See Also

PostGIS_Full_Version, PostGIS_Scripts_Released, PostGIS_Version

7.2.12 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
-------------------------
 1.3.4SVN
(1 row)
```

See Also

PostGIS_Full_Version, PostGIS_Scripts_Installed, PostGIS_Lib_Version

7.2.13 PostGIS_Uses_Stats

PostGIS_Uses_Stats — Returns `TRUE` if STATS usage has been enabled.

Synopsis

text **PostGIS_Uses_Stats**();

Description

Returns `TRUE` if STATS usage has been enabled, `FALSE` otherwise.

Examples

```
SELECT PostGIS_Uses_Stats();
 postgis_uses_stats
--------------------
 t
(1 row)
```

See Also

PostGIS_Version

7.2.14 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
---------------------------------------
 1.3 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

7.2.15 Populate_Geometry_Columns

Populate_Geometry_Columns — Ensures geometry columns have appropriate spatial constraints and exist in the `geometry_columns` table.

Synopsis

text **Populate_Geometry_Columns**();
int **Populate_Geometry_Columns**(oid relation_oid);

Description

Ensures geometry columns have appropriate spatial constraints and exist in the `geometry_columns` table. In particular, this means that every geometry column belonging to a table has at least three constraints:

- `enforce_dims_the_geom` - ensures every geometry has the same dimension (see ST_NDims)
- `enforce_geotype_the_geom` - ensures every geometry is of the same type (see GeometryType)
- `enforce_srid_the_geom` - ensures every geometry is in the same projection (see ST_SRID)

If a table `oid` is provided, this function tries to determine the srid, dimension, and geometry type of all geometry columns in the table, adding contraints as necessary. If successful, an appropriate row is inserted into the geometry_columns table, otherwise, the exception is caught and an error notice is raised describing the problem.

If the `oid` of a view is provided, as with a table oid, this function tries to determine the srid, dimension, and type of all the geometries in the view, inserting appropriate entries into the `geometry_columns` table, but nothing is done to enforce contraints.

The parameterless variant is a simple wrapper for the parameterized variant that first truncates and repopulates the geometry_columns table for every spatial table and view in the database, adding spatial contraints to tables where appropriate. It returns a summary of the number of geometry columns detected in the database and the number that were inserted into the `geometry_columns` table. The parameterized version simply returns the number of rows inserted into the `geometry_columns` table.

Availability: 1.4.0

Examples

```
SELECT Populate_Geometry_Columns('public.myspatial_table'::regclass);
```

See Also

Probe_Geometry_Columns

7.2.16 Probe_Geometry_Columns

Probe_Geometry_Columns — Scans all tables with PostGIS geometry constraints and adds them to the `geometry_columns` table if they are not there.

Synopsis

text **Probe_Geometry_Columns**();

Description

Scans all tables with PostGIS geometry constraints and adds them to the `geometry_columns` table if they are not there. Also give stats on number of inserts and already present or possibly obsolete.

Note
This will usually only pick up records added by AddGeometryColumn() function. It will not scan views so views will need to be manually added to geometry_columns table.

Examples

```
SELECT Probe_Geometry_Columns();
      probe_geometry_columns
---------------------------------------
probed:6 inserted:0 conflicts:6 stale:0
(1 row)
```

See Also

AddGeometryColumn

7.2.17 UpdateGeometrySRID

UpdateGeometrySRID — Updates the SRID of all features in a geometry column, geometry_columns metadata and srid table constraint

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);

Description

Updates the SRID of all features in a geometry column, updating constraints and reference in geometry_columns. Note: uses current_schema() on schema-aware pgsql installations if schema is not provided.

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

✓ This method supports Circular Strings and Curves

See Also

ST_SetSRID

7.3 Geometry Constructors

7.3.1 ST_BdPolyFromText

ST_BdPolyFromText — Construct a Polygon given an arbitrary collection of closed linestrings as a MultiLineString Well-Known text representation.

Synopsis

geometry **ST_BdPolyFromText**(text WKT, integer srid);

Description

Construct a Polygon given an arbitrary collection of closed linestrings as a MultiLineString Well-Known text representation.

Note
Throws an error if WKT is not a MULTILINESTRING. Throws an error if output is a MULTIPOLYGON; use ST_BdMPolyFromText in that case, or see ST_BuildArea() for a postgis-specific approach.

This method implements the OpenGIS Simple Features Implementation Specification for SQL 1.1. s3.2.6.2

Availability: 1.1.0 - requires GEOS >= 2.1.0.

Examples

```
Forthcoming
```

See Also

ST_BuildArea, ST_BdMPolyFromText

7.3.2 ST_BdMPolyFromText

ST_BdMPolyFromText — Construct a MultiPolygon given an arbitrary collection of closed linestrings as a MultiLineString text representation Well-Known text representation.

Synopsis

geometry **ST_BdMPolyFromText**(text WKT, integer srid);

Description

Construct a Polygon given an arbitrary collection of closed linestrings, polygons, MultiLineStrings as Well-Known text representation.

Note
Throws an error if WKT is not a MULTILINESTRING. Forces MULTIPOLYGON output even when result is really only composed by a single POLYGON; use ST_BdPolyFromText if you're sure a single POLYGON will result from operation, or see ST_BuildArea() for a postgis-specific approach.

This method implements the OpenGIS Simple Features Implementation Specification for SQL 1.1. s3.2.6.2

Availability: 1.1.0 - requires GEOS >= 2.1.0.

Examples

```
Forthcoming
```

See Also

ST_BuildArea, ST_BdPolyFromText

7.3.3 ST_GeogFromText

ST_GeogFromText — Return a specified geography value from Well-Known Text representation or extended (WKT).

Synopsis

geography **ST_GeogFromText**(text EWKT);

Description

Returns a geography object from the well-known text or extended well-known representation. SRID 4326 is assumed. This is an alias for ST_GeographyFromText

Examples

```
--- 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 || ')') ↩
    ;
```

See Also

ST_AsText,ST_GeographyFromText

7.3.4 ST_GeographyFromText

ST_GeographyFromText — Return a specified geography value from Well-Known Text representation or extended (WKT).

Synopsis

geography **ST_GeographyFromText**(text EWKT);

Description

Returns a geography object from the well-known text representation. SRID 4326 is assumed.

See Also

ST_AsText

7.3.5 ST_GeogFromWKB

ST_GeogFromWKB — Creates a geography instance from a Well-Known Binary geometry representation (WKB) or extended Well Known Binary (EWKB).

Synopsis

geography **ST_GeogFromWKB**(bytea geom);

Description

The `ST_GeogFromWKB` function, takes a well-known binary representation (WKB) of a geometry or PostGIS Extended WKB and creates an instance of the appropriate geography type. This function plays the role of the Geometry Factory in SQL.

If SRID is not specified, it defaults to 4326 (WGS 84 long lat).

✓ This method supports Circular Strings and Curves

Examples

```
--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)
```

See Also

ST_GeogFromText, ST_AsBinary

7.3.6 ST_GeomCollFromText

ST_GeomCollFromText — Makes a collection Geometry from collection WKT with the given SRID. If SRID is not give, it defaults to -1.

Synopsis

geometry **ST_GeomCollFromText**(text WKT, integer srid);
geometry **ST_GeomCollFromText**(text WKT);

Description

Makes a collection Geometry from the Well-Known-Text (WKT) representation with the given SRID. If SRID is not give, it defaults to -1.

OGC SPEC 3.2.6.2 - option SRID is from the conformance suite

Returns null if the WKT is not a GEOMETRYCOLLECTION

Note
If you are absolutely sure all your WKT geometries are collections, don't use this function. It is slower than ST_GeomFromText since it adds an additional validation step.

✓ This method implements the OpenGIS Simple Features Implementation Specification for SQL 1.1. s3.2.6.2

✓ This method implements the SQL/MM specification.

Examples

```
SELECT ST_GeomCollFromText('GEOMETRYCOLLECTION(POINT(1 2),LINESTRING(1 2, 3 4))');
```

See Also

ST_GeomFromText, ST_SRID

7.3.7 ST_GeomFromEWKB

ST_GeomFromEWKB — Return a specified ST_Geometry value from Extended Well-Known Binary representation (EWKB).

Synopsis

geometry **ST_GeomFromEWKB**(bytea EWKB);

Description

Constructs a PostGIS ST_Geometry object from the OGC Extended Well-Known binary (EWKT) representation.

Note
The EWKB format is not an OGC standard, but a PostGIS specific format that includes the spatial reference system (SRID) identifier

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

✓ This method supports Circular Strings and Curves

Examples

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
NOTE: Even though byte arrays are delimited with \ and may have ', we need to escape both out with \ and ''. So it does not look exactly like its AsEWKB representation.

```
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@');
```

See Also

ST_AsBinary, ST_AsEWKB, ST_GeomFromWKB

7.3.8 ST_GeomFromEWKT

ST_GeomFromEWKT — Return a specified ST_Geometry value from Extended Well-Known Text representation (EWKT).

Synopsis

geometry **ST_GeomFromEWKT**(text EWKT);

Description

Constructs a PostGIS ST_Geometry object from the OGC Extended Well-Known text (EWKT) representation.

Note
The EWKT format is not an OGC standard, but an PostGIS specific format that includes the spatial reference system (SRID) identifier

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

✓ This method supports Circular Strings and Curves

Examples

```
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)’);
```

See Also

ST_AsEWKT, ST_GeomFromText, ST_GeomFromEWKT

7.3.9 ST_GeometryFromText

ST_GeometryFromText — Return a specified ST_Geometry value from Well-Known Text representation (WKT). This is an alias name for ST_GeomFromText

Synopsis

geometry **ST_GeometryFromText**(text WKT);
geometry **ST_GeometryFromText**(text WKT, integer srid);

Description

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

✓ This method implements the SQL/MM specification. SQL-MM 3: 5.1.40

See Also

ST_GeomFromText

7.3.10 ST_GeomFromGML

ST_GeomFromGML — Takes as input GML representation of geometry and outputs a PostGIS geometry object

Synopsis

geometry **ST_GeomFromGML**(text geomgml);

Description

Constructs a PostGIS ST_Geometry object from the OGC GML representation.

ST_GeomFromGML works only for GML Geometry fragments. It throws an error if you try to use it on a whole GML document.

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:

Availability: 1.5

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

GML allow mixed dimensions (2D and 3D inside the same MultiGeometry for instance). As PostGIS geometries don't, ST_GeomFromGML convert the whole geometry to 2D if a missing Z dimension is found once.

GML support mixed SRS inside the same MultiGeometry. As PostGIS geometries don't, ST_GeomFromGML, in this case, reproject all subgeometries to the SRS root node. If no srsName attribute available for the GML root node, the function throw an error.

ST_GeomFromGML function is not pedantic about an explicit GML namespace. You could avoid to mention it explicitly for common usages. But you need it if you want to use XLink feature inside GML.

Note
ST_GeomFromGML function not support SQL/MM curves geometries.

Examples - A single geometry with srsName

```
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>');
```

Examples - XLink usage

```
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>'););
```

See Also

ST_AsGML

ST_GMLToSQL

7.3.11 ST_GeomFromKML

ST_GeomFromKML — Takes as input KML representation of geometry and outputs a PostGIS geometry object

Synopsis

geometry **ST_GeomFromKML**(text geomkml);

Description

Constructs a PostGIS ST_Geometry object from the OGC KML representation.

ST_GeomFromKML works only for KML Geometry fragments. It throws an error if you try to use it on a whole KML document.

OGC KML versions supported:

- KML 2.2.0 Namespace

OGC KML standards, cf: http://www.opengeospatial.org/standards/kml:

Availability: 1.5

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

Note
ST_GeomFromKML function not support SQL/MM curves geometries.

Examples - A single geometry with srsName

```
SELECT ST_GeomFromKML(’
    <LineString>
      <coordinates>-71.1663,42.2614
        -71.1667,42.2616</coordinates>
    </LineString>’);
```

See Also

ST_AsKML

7.3.12 ST_GMLToSQL

ST_GMLToSQL — Return a specified ST_Geometry value from GML representation. This is an alias name for ST_GeomFromGML

Synopsis

geometry **ST_GMLToSQL**(text geomgml);

Description

This method implements the SQL/MM specification. SQL-MM 3: 5.1.50 (except for curves support).

Availability: 1.5

See Also

ST_GeomFromGML

ST_AsGML

7.3.13 ST_GeomFromText

ST_GeomFromText — Return a specified ST_Geometry value from Well-Known Text representation (WKT).

Synopsis

geometry **ST_GeomFromText**(text WKT);
geometry **ST_GeomFromText**(text WKT, integer srid);

Description

Constructs a PostGIS ST_Geometry object from the OGC Well-Known text representation.

> **Note**
> There are 2 variants of ST_GeomFromText function, the first takes no SRID and returns a geometry with no defined spatial reference system. The second takes a spatial reference id as the second argument and returns an ST_Geometry that includes this srid as part of its meta-data. The srid must be defined in the spatial_ref_sys table.

This method implements the OpenGIS Simple Features Implementation Specification for SQL 1.1. s3.2.6.2 - option SRID is from the conformance suite.

This method implements the SQL/MM specification. SQL-MM 3: 5.1.40

This method supports Circular Strings and Curves

Examples

```
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)');
```

See Also

ST_GeomFromEWKT, ST_GeomFromWKB, ST_SRID

7.3.14 ST_GeomFromWKB

ST_GeomFromWKB — Creates a geometry instance from a Well-Known Binary geometry representation (WKB) and optional SRID.

Synopsis

geometry **ST_GeomFromWKB**(bytea geom);
geometry **ST_GeomFromWKB**(bytea geom, integer srid);

Description

The `ST_GeomFromWKB` function, takes a well-known binary representation of a geometry and a Spatial Reference System ID (`SRID`) and creates an instance of the appropriate geometry type. This function plays the role of the Geometry Factory in SQL. This is an alternate name for ST_WKBToSQL.

If SRID is not specified, it defaults to -1 (Unknown).

This method implements the OpenGIS Simple Features Implementation Specification for SQL 1.1. s3.2.7.2 - the optional SRID is from the conformance suite

This method implements the SQL/MM specification. SQL-MM 3: 5.1.41

This method supports Circular Strings and Curves

Examples

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

See Also

ST_WKBToSQL, ST_AsBinary, ST_GeomFromEWKB

7.3.15 ST_LineFromMultiPoint

ST_LineFromMultiPoint — Creates a LineString from a MultiPoint geometry.

Synopsis

geometry **ST_LineFromMultiPoint**(geometry aMultiPoint);

Description

Creates a LineString from a MultiPoint geometry.

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

Examples

```
--Create a 3d line string from a 3d multipoint
SELECT ST_AsEWKT(ST_LineFromMultiPoint(ST_GeomFromEWKT('MULTIPOINT(1 2 3, 4 5 6, 7 8 9)'))) ↩
    ;
--result--
LINESTRING(1 2 3,4 5 6,7 8 9)
```

See Also

ST_AsEWKT, ST_Collect,ST_MakeLine

7.3.16 ST_LineFromText

ST_LineFromText — Makes a Geometry from WKT representation with the given SRID. If SRID is not given, it defaults to -1.

Synopsis

geometry **ST_LineFromText**(text WKT);
geometry **ST_LineFromText**(text WKT, integer srid);

Description

Makes a Geometry from WKT with the given SRID. If SRID is not give, it defaults to -1. If WKT passed in is not a LINESTRING, then null is returned.

Note
OGC SPEC 3.2.6.2 - option SRID is from the conformance suite.

Note
If you know all your geometries are LINESTRINGS, its more efficient to just use ST_GeomFromText. This just calls ST_GeomFromText and adds additional validation that it returns a linestring.

This method implements the OpenGIS 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

Examples

```
SELECT ST_LineFromText('LINESTRING(1 2, 3 4)') AS aline, ST_LineFromText('POINT(1 2)') AS  ↩
    null_return;
aline                            | null_return
------------------------------------------------
010200000002000000000000000000F ... | t
```

See Also

ST_GeomFromText

7.3.17 ST_LineFromWKB

ST_LineFromWKB — Makes a `LINESTRING` from WKB with the given SRID

Synopsis

geometry **ST_LineFromWKB**(bytea WKB);
geometry **ST_LineFromWKB**(bytea WKB, integer srid);

Description

The `ST_LineFromWKB` function, takes a well-known binary representation of geometry and a Spatial Reference System ID (`SRID`) and creates an instance of the appropriate geometry type - in this case, a `LINESTRING` geometry. This function plays the role of the Geometry Factory in SQL.

If an SRID is not specified, it defaults to -1. `NULL` is returned if the input `bytea` does not represent a `LINESTRING`.

Note
OGC SPEC 3.2.6.2 - option SRID is from the conformance suite.

Note
If you know all your geometries are `LINESTRING`s, its more efficient to just use ST_GeomFromWKB. This function just calls ST_GeomFromWKB and adds additional validation that it returns a linestring.

✓ This method implements the OpenGIS 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

Examples

```
SELECT ST_LineFromWKB(ST_AsBinary(ST_GeomFromText('LINESTRING(1 2, 3 4)'))) AS aline,
    ST_LineFromWKB(ST_AsBinary(ST_GeomFromText('POINT(1 2)'))) IS NULL AS null_return;
aline                            | null_return
------------------------------------------------
010200000002000000000000000000F ... | t
```

See Also

ST_GeomFromWKB, ST_LinestringFromWKB

7.3.18 ST_LinestringFromWKB

ST_LinestringFromWKB — Makes a geometry from WKB with the given SRID.

Synopsis

geometry **ST_LinestringFromWKB**(bytea WKB);
geometry **ST_LinestringFromWKB**(bytea WKB, integer srid);

Description

The `ST_LinestringFromWKB` function, takes a well-known binary representation of geometry and a Spatial Reference System ID (`SRID`) and creates an instance of the appropriate geometry type - in this case, a `LINESTRING` geometry. This function plays the role of the Geometry Factory in SQL.

If an SRID is not specified, it defaults to -1. `NULL` is returned if the input `bytea` does not represent a `LINESTRING` geometry. This an alias for ST_LineFromWKB.

Note
OGC SPEC 3.2.6.2 - optional SRID is from the conformance suite.

Note
If you know all your geometries are `LINESTRING`s, it's more efficient to just use ST_GeomFromWKB. This function just calls ST_GeomFromWKB and adds additional validation that it returns a `LINESTRING`.

✓ This method implements the OpenGIS 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

Examples

```
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
------------------------------------------------
0102000000020000000000000000000F ... | t
```

See Also

ST_GeomFromWKB, ST_LineFromWKB

7.3.19 ST_MakeBox2D

ST_MakeBox2D — Creates a BOX2D defined by the given point geometries.

Synopsis

box2d **ST_MakeBox2D**(geometry pointLowLeft, geometry pointUpRight);

Description

Creates a BOX2D defined by the given 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, the_geom
FROM features
WHERE the_geom && ST_SetSRID(ST_MakeBox2D(ST_Point(-989502.1875, 528439.5625),
  ST_Point(-987121.375 ,529933.1875)),2163)
```

See Also

ST_MakePoint, ST_Point, ST_SetSRID, ST_SRID

7.3.20 ST_MakeBox3D

ST_MakeBox3D — Creates a BOX3D defined by the given 3d point geometries.

Synopsis

box3d **ST_MakeBox3D**(geometry point3DLowLeftBottom, geometry point3DUpRightTop);

Description

Creates a BOX3D defined by the given 2 3D point geometries.

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

Examples

```
SELECT ST_MakeBox3D(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

7.3.21 ST_MakeLine

ST_MakeLine — Creates a Linestring from point geometries.

Synopsis

geometry **ST_MakeLine**(geometry set pointfield);
geometry **ST_MakeLine**(geometry point1, geometry point2);
geometry **ST_MakeLine**(geometry[] point_array);

Description

ST_MakeLine comes in 3 forms: a spatial aggregate that takes rows of point geometries and returns a line string, a function that takes an array of points, and a regular function that takes two point geometries. You might want to use a subselect to order points before feeding them to the aggregate version of this function.

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

Availability: 1.4.0 - ST_MakeLine(geomarray) was introduced. ST_MakeLine aggregate functions was enhanced to handle more points faster.

Examples: Spatial Aggregate version

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

```
-- For pre-PostgreSQL 9.0 - this usually works,
-- but the planner may on occasion choose not to respect the order of the subquery
SELECT gps.gps_track, ST_MakeLine(gps.the_geom) As newgeom
  FROM (SELECT gps_track,gps_time, the_geom
      FROM gps_points ORDER BY gps_track, gps_time) As gps
  GROUP BY gps.gps_track;
```

```
-- If you are using PostgreSQL 9.0+
-- (you can use the new ORDER BY support for aggregates)
-- this is a guaranteed way to get a correctly ordered linestring
-- Your order by part can order by more than one column if needed
SELECT gps.gps_track, ST_MakeLine(gps.the_geom ORDER BY gps_time) As newgeom
  FROM gps_points As gps
  GROUP BY gps.gps_track;
```

Examples: Non-Spatial Aggregate version

First example is a simple one off line string composed of 2 points. The second formulates line strings from 2 points a user draws. The third is a one-off that joins 2 3d points to create a line in 3d space.

```
SELECT ST_AsText(ST_MakeLine(ST_MakePoint(1,2), ST_MakePoint(3,4)));
    st_astext
---------------------
 LINESTRING(1 2,3 4)

SELECT userpoints.id, ST_MakeLine(startpoint, endpoint) As drawn_line
  FROM userpoints ;

SELECT ST_AsEWKT(ST_MakeLine(ST_MakePoint(1,2,3), ST_MakePoint(3,4,5)));
    st_asewkt
-------------------------
 LINESTRING(1 2 3,3 4 5)
```

Examples: Using Array version

```
SELECT ST_MakeLine(ARRAY(SELECT ST_Centroid(the_geom) FROM visit_locations ORDER BY  ↩
    visit_time));

--Making a 3d line with 3 3-d points
SELECT ST_AsEWKT(ST_MakeLine(ARRAY[ST_MakePoint(1,2,3),
        ST_MakePoint(3,4,5), ST_MakePoint(6,6,6)]));
    st_asewkt
-------------------------
LINESTRING(1 2 3,3 4 5,6 6 6)
```

See Also

ST_AsEWKT, ST_AsText, ST_GeomFromText, ST_MakePoint

7.3.22 ST_MakeEnvelope

ST_MakeEnvelope — Creates a rectangular Polygon formed from the given minimums and maximums. Input values must be in SRS specified by the SRID.

Synopsis

geometry **ST_MakeEnvelope**(double precision xmin, double precision ymin, double precision xmax, double precision ymax, integer srid);

Description

Creates a rectangular Polygon formed from the minima and maxima. by the given shell. Input values must be in SRS specified by the SRID.

Availability: 1.5

Example: Building a bounding box polygon

```
SELECT ST_AsText(ST_MakeEnvelope(10, 10, 11, 11, 4326));

st_asewkt
-----------
POLYGON((10 10, 10 11, 11 11, 11 10, 10 10))
```

See Also

ST_MakePoint, ST_MakeLine, ST_MakePolygon

7.3.23 ST_MakePolygon

ST_MakePolygon — Creates a Polygon formed by the given shell. Input geometries must be closed LINESTRINGS.

Synopsis

geometry **ST_MakePolygon**(geometry linestring);

geometry **ST_MakePolygon**(geometry outerlinestring, geometry[] interiorlinestrings);

Description

Creates a Polygon formed by the given shell. Input geometries must be closed LINESTRINGS. Comes in 2 variants.

Variant 1: takes one closed linestring.

Variant 2: Creates a Polygon formed by the given shell and array of holes. You can construct a geometry array using ST_Accum or the PostgreSQL ARRAY[] and ARRAY() constructs. Input geometries must be closed LINESTRINGS.

> **Note**
> This function will not accept a MULTILINESTRING. Use ST_LineMerge or ST_Dump to generate line strings.

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

Examples: Single closed LINESTRING

```
--2d line
SELECT ST_MakePolygon(ST_GeomFromText('LINESTRING(75.15 29.53,77 29,77.6 29.5, 75.15 29.53) ↩
    '));
--If linestring is not closed
--you can add the start point to close it
SELECT ST_MakePolygon(ST_AddPoint(foo.open_line, ST_StartPoint(foo.open_line)))
FROM (
SELECT ST_GeomFromText('LINESTRING(75.15 29.53,77 29,77.6 29.5)') As open_line) As foo;

--3d closed line
SELECT ST_MakePolygon(ST_GeomFromText('LINESTRING(75.15 29.53 1,77 29 1,77.6 29.5 1, 75.15  ↩
    29.53 1)'));

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

--measured line --
SELECT ST_MakePolygon(ST_GeomFromText('LINESTRINGM(75.15 29.53 1,77 29 1,77.6 29.5 2, 75.15 ↩
     29.53 2)'));

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

Examples: Outter shell with inner shells

Build a donut with an ant hole

```
SELECT ST_MakePolygon(
    ST_ExteriorRing(ST_Buffer(foo.line,10)),
  ARRAY[ST_Translate(foo.line,1,1),
    ST_ExteriorRing(ST_Buffer(ST_MakePoint(20,20),1)) ]
  )
FROM
  (SELECT ST_ExteriorRing(ST_Buffer(ST_MakePoint(10,10),10,10))
    As line )
    As foo;
```

Build province boundaries with holes representing lakes in the province from a set of province polygons/multipolygons and water line strings this is an example of using PostGIS ST_Accum

Note
The use of CASE because feeding a null array into ST_MakePolygon results in NULL

Note
the use of left join to guarantee we get all provinces back even if they have no lakes

```
SELECT p.gid, p.province_name,
  CASE WHEN
    ST_Accum(w.the_geom) IS NULL THEN p.the_geom
  ELSE  ST_MakePolygon(ST_LineMerge(ST_Boundary(p.the_geom)), ST_Accum(w.the_geom)) END
FROM
  provinces p LEFT JOIN waterlines w
    ON (ST_Within(w.the_geom, p.the_geom) AND ST_IsClosed(w.the_geom))
GROUP BY p.gid, p.province_name, p.the_geom;

--Same example above but utilizing a correlated subquery
--and PostgreSQL built-in ARRAY() function that converts a row set to an array

SELECT p.gid,  p.province_name, CASE WHEN
  EXISTS(SELECT w.the_geom
    FROM waterlines w
    WHERE ST_Within(w.the_geom, p.the_geom)
    AND ST_IsClosed(w.the_geom))
  THEN
  ST_MakePolygon(ST_LineMerge(ST_Boundary(p.the_geom)),
    ARRAY(SELECT w.the_geom
      FROM waterlines w
      WHERE ST_Within(w.the_geom, p.the_geom)
      AND ST_IsClosed(w.the_geom)))
  ELSE p.the_geom END As the_geom
FROM
  provinces p;
```

See Also

ST_Accum, ST_AddPoint, ST_GeometryType, ST_IsClosed, ST_LineMerge

7.3.24 ST_MakePoint

ST_MakePoint — Creates a 2D,3DZ or 4D point geometry.

Synopsis

geometry **ST_MakePoint**(double precision x, double precision y);

geometry **ST_MakePoint**(double precision x, double precision y, double precision z);

geometry **ST_MakePoint**(double precision x, double precision y, double precision z, double precision m);

Description

Creates a 2D,3DZ or 4D point geometry (geometry with measure). `ST_MakePoint` while not being OGC compliant is generally faster and more precise than ST_GeomFromText and ST_PointFromText. It is also easier to use if you have raw coordinates rather than WKT.

> **Note**
> Note x is longitude and y is latitude

> **Note**
> Use ST_MakePointM if you need to make a point with x,y,m.

✓ 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
```

See Also

ST_GeomFromText, ST_PointFromText, ST_SetSRID, ST_MakePointM

7.3.25 ST_MakePointM

ST_MakePointM — Creates a point geometry with an x y and m coordinate.

Synopsis

geometry **ST_MakePointM**(float x, float y, float m);

Description

Creates a point with x, y and measure coordinates.

Note
Note x is longitude and y is latitude.

Examples

We use ST_AsEWKT in these examples to show the text representation instead of ST_AsText because ST_AsText does not support returning M.

```
--Return EWKT representation of point with unknown SRID
SELECT ST_AsEWKT(ST_MakePointM(-71.1043443253471, 42.3150676015829, 10));

--result
           st_asewkt
-----------------------------------------------
 POINTM(-71.1043443253471 42.3150676015829 10)

--Return EWKT representation of point with measure marked as WGS 84 long lat
SELECT ST_AsEWKT(ST_SetSRID(ST_MakePointM(-71.1043443253471, 42.3150676015829,10),4326));

            st_asewkt
---------------------------------------------------------
SRID=4326;POINTM(-71.1043443253471 42.3150676015829 10)

--Return a 3d point (e.g. has altitude)
SELECT ST_MakePoint(1, 2,1.5);

--Get m of point
SELECT ST_M(ST_MakePointM(-71.1043443253471, 42.3150676015829,10));
result
-------
10
```

See Also

ST_AsEWKT, ST_MakePoint, ST_SetSRID

7.3.26 ST_MLineFromText

ST_MLineFromText — Return a specified ST_MultiLineString value from WKT representation.

Synopsis

geometry **ST_MLineFromText**(text WKT, integer srid);
geometry **ST_MLineFromText**(text WKT);

Description

Makes a Geometry from Well-Known-Text (WKT) with the given SRID. If SRID is not give, it defaults to -1.

OGC SPEC 3.2.6.2 - option SRID is from the conformance suite

Returns null if the WKT is not a MULTILINESTRING

Note
If you are absolutely sure all your WKT geometries are points, don't use this function. It is slower than ST_GeomFromText since it adds an additional validation step.

This method implements the OpenGIS 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

Examples

```
SELECT ST_MLineFromText('MULTILINESTRING((1 2, 3 4), (4 5, 6 7))');
```

See Also

ST_GeomFromText

7.3.27 ST_MPointFromText

ST_MPointFromText — Makes a Geometry from WKT with the given SRID. If SRID is not give, it defaults to -1.

Synopsis

geometry **ST_MPointFromText**(text WKT, integer srid);
geometry **ST_MPointFromText**(text WKT);

Description

Makes a Geometry from WKT with the given SRID. If SRID is not give, it defaults to -1.

OGC SPEC 3.2.6.2 - option SRID is from the conformance suite

Returns null if the WKT is not a MULTIPOINT

Note
If you are absolutely sure all your WKT geometries are points, don't use this function. It is slower than ST_GeomFromText since it adds an additional validation step.

This method implements the OpenGIS 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

Examples

```
SELECT ST_MPointFromText(’MULTIPOINT(1 2, 3 4)’);
SELECT ST_MPointFromText(’MULTIPOINT(-70.9590 42.1180, -70.9611 42.1223)’, 4326);
```

See Also

ST_GeomFromText

7.3.28 ST_MPolyFromText

ST_MPolyFromText — Makes a MultiPolygon Geometry from WKT with the given SRID. If SRID is not give, it defaults to -1.

Synopsis

geometry **ST_MPolyFromText**(text WKT, integer srid);
geometry **ST_MPolyFromText**(text WKT);

Description

Makes a MultiPolygon from WKT with the given SRID. If SRID is not give, it defaults to -1.

OGC SPEC 3.2.6.2 - option SRID is from the conformance suite

Throws an error if the WKT is not a MULTIPOLYGON

Note
If you are absolutely sure all your WKT geometries are multipolygons, don’t use this function. It is slower than ST_GeomFromText since it adds an additional validation step.

✓ This method implements the OpenGIS 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

Examples

```
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);
```

See Also

ST_GeomFromText, ST_SRID

7.3.29 ST_Point

ST_Point — Returns an ST_Point with the given coordinate values. OGC alias for ST_MakePoint.

Synopsis

geometry **ST_Point**(float x_lon, float y_lat);

Description

Returns an ST_Point with the given coordinate values. MM compliant alias for ST_MakePoint that takes just an x and y.

✓ This method implements the SQL/MM specification. SQL-MM 3: 6.1.2

Examples: Geometry

```
SELECT ST_SetSRID(ST_Point(-71.1043443253471, 42.3150676015829),4326)
```

Examples: Geography

```
SELECT CAST(ST_SetSRID(ST_Point(-71.1043443253471, 42.3150676015829),4326) As geography);
```

```
-- the :: is PostgreSQL short-hand for casting.
SELECT ST_SetSRID(ST_Point(-71.1043443253471, 42.3150676015829),4326)::geography;
```

```
--If your point coordinates are in a different spatial reference from WGS-84 long lat, then ↩
     you need to transform before casting
-- This example we convert a point in Pennsylvania State Plane feet to WGS 84 and then  ↩
    geography
SELECT ST_Transform(ST_SetSRID(ST_Point(3637510, 3014852),2273),4326)::geography
```

;

See Also

Section 4.2.1, ST_MakePoint, ST_SetSRID, ST_Transform

7.3.30 ST_PointFromText

ST_PointFromText — Makes a point Geometry from WKT with the given SRID. If SRID is not given, it defaults to unknown.

Synopsis

geometry **ST_PointFromText**(text WKT);
geometry **ST_PointFromText**(text WKT, integer srid);

Description

Constructs a PostGIS ST_Geometry point object from the OGC Well-Known text representation. If SRID is not give, it defaults to unknown (currently -1). If geometry is not a WKT point representation, returns null. If completely invalid WKT, then throws an error.

Note
There are 2 variants of ST_PointFromText function, the first takes no SRID and returns a geometry with no defined spatial reference system. The second takes a spatial reference id as the second argument and returns an ST_Geometry that includes this srid as part of its meta-data. The srid must be defined in the spatial_ref_sys table.

Note
If you are absolutely sure all your WKT geometries are points, don't use this function. It is slower than ST_GeomFromText since it adds an additional validation step. If you are building points from long lat coordinates and care more about performance and accuracy than OGC compliance, use ST_MakePoint or OGC compliant alias ST_Point.

✓ This method implements the OpenGIS Simple Features Implementation Specification for SQL 1.1. s3.2.6.2 - option SRID is from the conformance suite.

✓ This method implements the SQL/MM specification. SQL-MM 3: 6.1.8

Examples

```
SELECT ST_PointFromText('POINT(-71.064544 42.28787)');
SELECT ST_PointFromText('POINT(-71.064544 42.28787)', 4326);
```

See Also

ST_GeomFromText, ST_MakePoint, ST_Point, ST_SRID

7.3.31 ST_PointFromWKB

ST_PointFromWKB — Makes a geometry from WKB with the given SRID

Synopsis

geometry **ST_GeomFromWKB**(bytea geom);
geometry **ST_GeomFromWKB**(bytea geom, integer srid);

Description

The `ST_PointFromWKB` function, takes a well-known binary representation of geometry and a Spatial Reference System ID (`SRID`) and creates an instance of the appropriate geometry type - in this case, a `POINT` geometry. This function plays the role of the Geometry Factory in SQL.

If an SRID is not specified, it defaults to -1. `NULL` is returned if the input `bytea` does not represent a `POINT` geometry.

✓ This method implements the OpenGIS 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

Examples

```
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)
```

See Also

ST_GeomFromWKB, ST_LineFromWKB

7.3.32 ST_Polygon

ST_Polygon — Returns a polygon built from the specified linestring and SRID.

Synopsis

geometry **ST_Polygon**(geometry aLineString, integer srid);

Description

Returns a polygon built from the specified linestring and SRID.

Note

ST_Polygon is similar to first version oST_MakePolygon except it also sets the spatial ref sys (SRID) of the polygon. Will not work with MULTILINESTRINGS so use LineMerge to merge multilines. Also does not create polygons with holes. Use ST_MakePolygon for that.

✓ This method implements the OpenGIS 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

```
--a 2d polygon
SELECT ST_Polygon(ST_GeomFromText('LINESTRING(75.15 29.53,77 29,77.6 29.5, 75.15 29.53)'),  ↩
    4326);

--result--
POLYGON((75.15 29.53,77 29,77.6 29.5,75.15 29.53))
--a 3d polygon
SELECT ST_AsEWKT(ST_Polygon(ST_GeomFromEWKT('LINESTRING(75.15 29.53 1,77 29 1,77.6 29.5 1,  ↩
    75.15 29.53 1)'), 4326));

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

See Also

ST_AsEWKT, ST_AsText, ST_GeomFromEWKT, ST_GeomFromText, ST_LineMerge, ST_MakePolygon

7.3.33 ST_PolygonFromText

ST_PolygonFromText — Makes a Geometry from WKT with the given SRID. If SRID is not give, it defaults to -1.

Synopsis

geometry **ST_PolygonFromText**(text WKT);
geometry **ST_PolygonFromText**(text WKT, integer srid);

Description

Makes a Geometry from WKT with the given SRID. If SRID is not give, it defaults to -1. Returns null if WKT is not a polygon.

OGC SPEC 3.2.6.2 - option SRID is from the conformance suite

> **Note**
> If you are absolutely sure all your WKT geometries are polygons, don't use this function. It is slower than ST_GeomFromText since it adds an additional validation step.

✓ This method implements the OpenGIS 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

Examples

```
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
------------------
010300000001000000050000006...


SELECT ST_PolygonFromText('POINT(1 2)') IS NULL as point_is_notpoly;

point_is_not_poly
----------
t
```

See Also

ST_GeomFromText

7.3.34 ST_WKBToSQL

ST_WKBToSQL — Return a specified ST_Geometry value from Well-Known Binary representation (WKB). This is an alias name for ST_GeomFromWKB that takes no srid

Synopsis

geometry **ST_WKBToSQL**(bytea WKB);

Description

✓ This method implements the SQL/MM specification. SQL-MM 3: 5.1.36

See Also

ST_GeomFromWKB

7.3.35 ST_WKTToSQL

ST_WKTToSQL — Return a specified ST_Geometry value from Well-Known Text representation (WKT). This is an alias name for ST_GeomFromText

Synopsis

geometry **ST_WKTToSQL**(text WKT);

Description

✓ This method implements the SQL/MM specification. SQL-MM 3: 5.1.34

See Also

ST_GeomFromText

7.4 Geometry Accessors

7.4.1 GeometryType

GeometryType — Returns the type of the geometry as a string. Eg: 'LINESTRING', 'POLYGON', 'MULTIPOINT', etc.

Synopsis

text **GeometryType**(geometry geomA);

Description

Returns the type of the geometry as a string. Eg: 'LINESTRING', 'POLYGON', 'MULTIPOINT', etc.

OGC SPEC s2.1.1.1 - Returns the name of the instantiable subtype of Geometry of which this Geometry instance is a member. The name of the instantiable subtype of Geometry is returned as a string.

Note
This function also indicates if the geometry is measured, by returning a string of the form 'POINTM'.

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

✓ This method supports Circular Strings and Curves

Examples

```
SELECT GeometryType(ST_GeomFromText('LINESTRING(77.29 29.07,77.42 29.26,77.27 29.31,77.29  ↩
    29.07)'));
 geometrytype
--------------
 LINESTRING
```

See Also

ST_GeometryType

7.4.2 ST_Boundary

ST_Boundary — Returns the closure of the combinatorial boundary of this Geometry.

Synopsis

geometry **ST_Boundary**(geometry geomA);

Description

Returns the closure of the combinatorial boundary of this Geometry. The combinatorial boundary is defined as described in section 3.12.3.2 of the OGC SPEC. Because the result of this function is a closure, and hence topologically closed, the resulting boundary can be represented using representational geometry primitives as discussed in the OGC SPEC, section 3.12.2.

Performed by the GEOS module

> **Important**
> Do not call with a GEOMETRYCOLLECTION as an argument

✓ This method implements the OpenGIS Simple Features Implementation Specification for SQL 1.1. OGC SPEC s2.1.1.1

✓ This method implements the SQL/MM specification. SQL-MM 3: 5.1.14

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

Examples

```
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)
```

See Also

ST_ExteriorRing, ST_MakePolygon

7.4.3 ST_CoordDim

ST_CoordDim — Return the coordinate dimension of the ST_Geometry value.

Synopsis

integer **ST_CoordDim**(geometry geomA);

Description

Return the coordinate dimension of the ST_Geometry value.

This is the MM compliant alias name for ST_NDims

✓ This method implements the OpenGIS 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.

Examples

```
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
```

See Also

ST_NDims

7.4.4 ST_Dimension

ST_Dimension — The inherent dimension of this Geometry object, which must be less than or equal to the coordinate dimension.

Synopsis

integer **ST_Dimension**(geometry g);

Description

The inherent dimension of this Geometry object, which must be less than or equal to the coordinate dimension. OGC SPEC s2.1.1.1 - returns 0 for `POINT`, 1 for `LINESTRING`, 2 for `POLYGON`, and the largest dimension of the components of a `GEOMETRYCOLLECTION`.

✓ This method implements the SQL/MM specification. SQL-MM 3: 5.1.2

Examples

```
SELECT ST_Dimension('GEOMETRYCOLLECTION(LINESTRING(1 1,0 0),POINT(0 0))');
ST_Dimension
-----------
1
```

See Also

ST_NDims

7.4.5 ST_EndPoint

ST_EndPoint — Returns the last point of a `LINESTRING` geometry as a `POINT`.

Synopsis

boolean **ST_EndPoint**(geometry g);

Description

Returns the last point of a `LINESTRING` geometry as a `POINT` or `NULL` if the input parameter is not a `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.

Examples

```
postgis=# SELECT ST_AsText(ST_EndPoint('LINESTRING(1 1, 2 2, 3 3)'::geometry));
 st_astext
------------
 POINT(3 3)
(1 row)

postgis=# SELECT ST_EndPoint('POINT(1 1)'::geometry) IS NULL AS is_null;
  is_null
----------
 t
(1 row)

--3d endpoint
SELECT ST_AsEWKT(ST_EndPoint('LINESTRING(1 1 2, 1 2 3, 0 0 5)'));
  st_asewkt
--------------
 POINT(0 0 5)
(1 row)
```

See Also

ST_PointN, ST_StartPoint

7.4.6 ST_Envelope

ST_Envelope — Returns a geometry representing the double precision (float8) bounding box of the supplied geometry.

Synopsis

geometry **ST_Envelope**(geometry g1);

Description

Returns the float8 minimum bounding box for the supplied geometry, as a geometry. The polygon is defined by the corner points of the bounding box ((`MINX`, `MINY`), (`MINX`, `MAXY`), (`MAXX`, `MAXY`), (`MAXX`, `MINY`), (`MINX`, `MINY`)). (PostGIS will add a `ZMIN`/`ZMAX` coordinate as well).

Degenerate cases (vertical lines, points) will return a geometry of lower dimension than `POLYGON`, ie. `POINT` or `LINESTRING`.

Availability: 1.5.0 behavior changed to output double precision instead of float4

✓ This method implements the OpenGIS Simple Features Implementation Specification for SQL 1.1. s2.1.1.1

✓ This method implements the SQL/MM specification. SQL-MM 3: 5.1.15

Examples

```
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;
```

See Also

Box2D, Box3D

7.4.7 ST_ExteriorRing

ST_ExteriorRing — Returns a line string representing the exterior ring of the POLYGON geometry. Return NULL if the geometry is not a polygon. Will not work with MULTIPOLYGON

Synopsis

geometry **ST_ExteriorRing**(geometry a_polygon);

Description

Returns a line string representing the exterior ring of the POLYGON geometry. Return NULL if the geometry is not a polygon.

Note
Only works with POLYGON geometry types

✓ This method implements the OpenGIS 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.

Examples

```
--If you have a table of polygons
SELECT gid, ST_ExteriorRing(the_geom) AS ering
FROM sometable;

--If you have a table of MULTIPOLYGONs
--and want to return a MULTILINESTRING composed of the exterior rings of each polygon
SELECT gid, ST_Collect(ST_ExteriorRing(the_geom)) AS erings
  FROM (SELECT gid, (ST_Dump(the_geom)).geom As the_geom
      FROM sometable) As foo
GROUP BY gid;

--3d Example
SELECT ST_AsEWKT(
  ST_ExteriorRing(
  ST_GeomFromEWKT('POLYGON((0 0 1, 1 1 1, 1 2 1, 1 1 1, 0 0 1))')
  )
);

st_asewkt
---------
LINESTRING(0 0 1,1 1 1,1 2 1,1 1 1,0 0 1)
```

See Also

ST_Boundary, ST_NumInteriorRings

7.4.8 ST_GeometryN

ST_GeometryN — Return the 1-based Nth geometry if the geometry is a GEOMETRYCOLLECTION, MULTIPOINT, MULTILINESTRING, MULTICURVE or MULTIPOLYGON. Otherwise, return NULL.

Synopsis

geometry **ST_GeometryN**(geometry geomA, integer n);

Description

Return the 1-based Nth geometry if the geometry is a GEOMETRYCOLLECTION, MULTIPOINT, MULTILINESTRING, MULTICURVE or MULTIPOLYGON. Otherwise, return NULL.

Note
Index is 1-based as for OGC specs since version 0.8.0. Previous versions implemented this as 0-based instead.

Note
If you want to extract all geometries, of a geometry, ST_Dump is more efficient and will also work for singular geoms.

✓ This method implements the OpenGIS 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

Examples

```
--Extracting a subset of points from a 3d multipoint
SELECT n, ST_AsEWKT(ST_GeometryN(the_geom, n)) As geomewkt
FROM (
VALUES (ST_GeomFromEWKT('MULTIPOINT(1 2 7, 3 4 7, 5 6 7, 8 9 10)') ),
( ST_GeomFromEWKT('MULTICURVE(CIRCULARSTRING(2.5 2.5,4.5 2.5, 3.5 3.5), (10 11, 12 11))') )
  )As foo(the_geom)
  CROSS JOIN generate_series(1,100) n
WHERE n <= ST_NumGeometries(the_geom);

 n |                geomewkt
---+-----------------------------------------
 1 | POINT(1 2 7)
 2 | POINT(3 4 7)
 3 | POINT(5 6 7)
 4 | POINT(8 9 10)
 1 | CIRCULARSTRING(2.5 2.5,4.5 2.5,3.5 3.5)
 2 | LINESTRING(10 11,12 11)


--Extracting all geometries (useful when you want to assign an id)
SELECT gid, n, ST_GeometryN(the_geom, n)
FROM sometable CROSS JOIN generate_series(1,100) n
WHERE n <= ST_NumGeometries(the_geom);
```

See Also

ST_Dump, ST_NumGeometries

7.4.9 ST_GeometryType

ST_GeometryType — Return the geometry type of the ST_Geometry value.

Synopsis

text **ST_GeometryType**(geometry g1);

Description

Returns the type of the geometry as a string. EG: 'ST_Linestring', 'ST_Polygon','ST_MultiPolygon' etc. This function differs from GeometryType(geometry) in the case of the string and ST in front that is returned, as well as the fact that it will not indicate whether the geometry is measured.

✓ This method implements the SQL/MM specification. SQL-MM 3: 5.1.4

Examples

```
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
```

See Also

GeometryType

7.4.10 ST_InteriorRingN

ST_InteriorRingN — Return the Nth interior linestring ring of the polygon geometry. Return NULL if the geometry is not a polygon or the given N is out of range.

Synopsis

geometry **ST_InteriorRingN**(geometry a_polygon, integer n);

Description

Return the Nth interior linestring ring of the polygon geometry. Return NULL if the geometry is not a polygon or the given N is out of range. index starts at 1.

Note
This will not work for MULTIPOLYGONs. Use in conjunction with ST_Dump for MULTIPOLYGONS

✓ This method implements the OpenGIS 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.

Examples

```
SELECT ST_AsText(ST_InteriorRingN(the_geom, 1)) As the_geom
FROM (SELECT ST_BuildArea(
    ST_Collect(ST_Buffer(ST_Point(1,2), 20,3),
      ST_Buffer(ST_Point(1, 2), 10,3))) As the_geom
    )  as foo
```

See Also

ST_BuildArea, ST_Collect, ST_Dump, ST_NumInteriorRing, ST_NumInteriorRings

7.4.11 ST_IsClosed

ST_IsClosed — Returns `TRUE` if the `LINESTRING`'s start and end points are coincident.

Synopsis

boolean **ST_IsClosed**(geometry g);

Description

Returns `TRUE` if the `LINESTRING`'s start and end points are coincident.

✓ This method implements the OpenGIS 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 defines the result of `ST_IsClosed(NULL)` to be 0, while PostGIS returns `NULL`.

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

✓ This method supports Circular Strings and Curves

Examples

```
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)
```

See Also

ST_IsRing

7.4.12 ST_IsEmpty

ST_IsEmpty — Returns true if this Geometry is an empty geometry . If true, then this Geometry represents the empty point set - i.e. GEOMETRYCOLLECTION(EMPTY).

Synopsis

boolean **ST_IsEmpty**(geometry geomA);

Description

Returns true if this Geometry is an empty geometry . If true, then this Geometry represents an empty geometry collection, polygon, point etc.

Note
SQL-MM defines the result of ST_IsEmpty(NULL) to be 0, while PostGIS returns NULL.

✓ This method implements the OpenGIS 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

Examples

```
SELECT ST_IsEmpty(’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)
```

7.4.13 ST_IsRing

ST_IsRing — Returns `TRUE` if this `LINESTRING` is both closed and simple.

Synopsis

boolean **ST_IsRing**(geometry g);

Description

Returns TRUE if this LINESTRING is both ST_IsClosed (ST_StartPoint ((*g*)) ~= ST_Endpoint ((*g*))) and ST_IsSimple (does not self intersect).

✓ This method implements the OpenGIS 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 defines the result of ST_IsRing(NULL) to be 0, while PostGIS returns NULL.

Examples

```
SELECT ST_IsRing(the_geom), ST_IsClosed(the_geom), ST_IsSimple(the_geom)
FROM (SELECT ’LINESTRING(0 0, 0 1, 1 1, 1 0, 0 0)’::geometry AS the_geom) AS foo;
 st_isring | st_isclosed | st_issimple
-----------+-------------+-------------
 t         | t           | t
(1 row)

SELECT ST_IsRing(the_geom), ST_IsClosed(the_geom), ST_IsSimple(the_geom)
FROM (SELECT ’LINESTRING(0 0, 0 1, 1 0, 1 1, 0 0)’::geometry AS the_geom) AS foo;
 st_isring | st_isclosed | st_issimple
-----------+-------------+-------------
 f         | t           | f
(1 row)
```

See Also

ST_IsClosed, ST_IsSimple, ST_StartPoint, ST_EndPoint

7.4.14 ST_IsSimple

ST_IsSimple — Returns (TRUE) if this Geometry has no anomalous geometric points, such as self intersection or self tangency.

Synopsis

boolean **ST_IsSimple**(geometry geomA);

Description

Returns true if this Geometry has no anomalous geometric points, such as self intersection or self tangency. For more information on the OGC’s definition of geometry simplicity and validity, refer to "Ensuring OpenGIS compliancy of geometries"

Note
SQL-MM defines the result of ST_IsSimple(NULL) to be 0, while PostGIS returns NULL.

✓ This method implements the OpenGIS 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.

Examples

```
 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)
```

See Also

ST_IsValid

7.4.15 ST_IsValid

ST_IsValid — Returns `true` if the `ST_Geometry` is well formed.

Synopsis

boolean **ST_IsValid**(geometry g);

Description

Test if an ST_Geometry value is well formed. For geometries that are invalid, the PostgreSQL NOTICE will provide details of why it is not valid. For more information on the OGC's definition of geometry simplicity and validity, refer to "Ensuring OpenGIS compliancy of geometries"

Note
SQL-MM defines the result of ST_IsValid(NULL) to be 0, while PostGIS returns NULL.

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

✓ This method implements the SQL/MM specification. SQL-MM 3: 5.1.9

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_Summary

7.4.16 ST_IsValidReason

ST_IsValidReason — Returns text stating if a geometry is valid or not and if not valid, a reason why.

Synopsis

text **ST_IsValidReason**(geometry geomA);

Description

Returns text stating if a geometry is valid or not an if not valid, a reason why.

Useful in combination with ST_IsValid to generate a detailed report of invalid geometries and reasons.

Availability: 1.4 - requires GEOS >= 3.1.0.

Examples

```
--First 3 Rejects from a successful quintuplet experiment
SELECT gid, ST_IsValidReason(the_geom) as validity_info
FROM
(SELECT ST_MakePolygon(ST_ExteriorRing(e.buff), ST_Accum(f.line)) As the_geom, gid
FROM (SELECT ST_Buffer(ST_MakePoint(x1*10,y1), z1) As buff, x1*10 + y1*100 + z1*1000 As gid
  FROM generate_series(-4,6) x1
  CROSS JOIN generate_series(2,5) y1
  CROSS JOIN generate_series(1,8) z1
  WHERE x1 > y1*0.5 AND z1 < x1*y1) As e
  INNER JOIN (SELECT ST_Translate(ST_ExteriorRing(ST_Buffer(ST_MakePoint(x1*10,y1), z1)),y1 ↩
      *1, z1*2) As line
  FROM generate_series(-3,6) x1
  CROSS JOIN generate_series(2,5) y1
  CROSS JOIN generate_series(1,10) z1
  WHERE x1 > y1*0.75 AND z1 < x1*y1) As f
ON (ST_Area(e.buff) > 78 AND ST_Contains(e.buff, f.line))
GROUP BY gid, e.buff) As quintuplet_experiment
WHERE ST_IsValid(the_geom) = false
ORDER BY gid
LIMIT 3;

 gid  |      validity_info
------+--------------------------
 5330 | Self-intersection [32 5]
 5340 | Self-intersection [42 5]
 5350 | Self-intersection [52 5]

 --simple example
SELECT ST_IsValidReason('LINESTRING(220227 150406,2220227 150407,222020 150410)');

 st_isvalidreason
------------------
 Valid Geometry
```

See Also

ST_IsValid, ST_Summary

7.4.17 ST_M

ST_M — Return the M coordinate of the point, or NULL if not available. Input must be a point.

Synopsis

float **ST_M**(geometry a_point);

Description

Return the M coordinate of the point, or NULL if not available. Input must be a point.

Note
This is not (yet) part of the OGC spec, but is listed here to complete the point coordinate extractor function list.

✓ This method implements the OpenGIS 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.

Examples

```
SELECT ST_M(ST_GeomFromEWKT('POINT(1 2 3 4)'));
 st_m
------
   4
(1 row)
```

See Also

ST_GeomFromEWKT, ST_X, ST_Y, ST_Z

7.4.18 ST_NDims

ST_NDims — Returns coordinate dimension of the geometry as a small int. Values are: 2,3 or 4.

Synopsis

integer **ST_NDims**(geometry g1);

Description

Returns the coordinate dimension of the geometry. PostGIS supports 2 - (x,y) , 3 - (x,y,z) or 2D with measure - x,y,m, and 4 - 3D with measure space x,y,z,m

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

Examples

```
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
```

See Also

ST_CoordDim, ST_Dimension, ST_GeomFromEWKT

7.4.19 ST_NPoints

ST_NPoints — Return the number of points (vertexes) in a geometry.

Synopsis

integer **ST_NPoints**(geometry g1);

Description

Return the number of points in a geometry. Works for all geometries.

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

Examples

```
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
```

See Also

ST_NumPoints

7.4.20 ST_NRings

ST_NRings — If the geometry is a polygon or multi-polygon returns the number of rings.

Synopsis

integer **ST_NRings**(geometry geomA);

Description

If the geometry is a polygon or multi-polygon returns the number of rings. Unlike NumInteriorRings, it counts the outer rings as well.

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

✓ This method supports Circular Strings and Curves

Examples

```
SELECT ST_NRings(the_geom) As Nrings, ST_NumInteriorRings(the_geom) As ninterrings
          FROM (SELECT ST_GeomFromText('POLYGON((1 2, 3 4, 5 6, 1 2))') As the_geom) As foo ↩
              ;
   nrings | ninterrings
--------+-------------
    1 |           0
(1 row)
```

See Also

ST_NumInteriorRings

7.4.21 ST_NumGeometries

ST_NumGeometries — If geometry is a GEOMETRYCOLLECTION (or MULTI*) return the number of geometries, otherwise return NULL.

Synopsis

integer **ST_NumGeometries**(geometry a_multi_or_geomcollection);

Description

Returns the number of Geometries. If geometry is a GEOMETRYCOLLECTION (or MULTI*) return the number of geometries, otherwise return NULL.

✓ This method implements the SQL/MM specification. SQL-MM 3: 9.1.4

Examples

```
--Although ST_NumGeometries will return null when passed a single, you can wrap in ST_Multi ↩
     to force 1 or more for all geoms
SELECT ST_NumGeometries(ST_Multi(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
```

See Also

ST_GeometryN, ST_Multi

7.4.22 ST_NumInteriorRings

ST_NumInteriorRings — Return the number of interior rings of the first polygon in the geometry. This will work with both POLYGON and MULTIPOLYGON types but only looks at the first polygon. Return NULL if there is no polygon in the geometry.

Synopsis

integer **ST_NumInteriorRings**(geometry a_polygon);

Description

Return the number of interior rings of the first polygon in the geometry. This will work with both POLYGON and MULTIPOLYGON types but only looks at the first polygon. Return NULL if there is no polygon in the geometry.

✓ This method implements the SQL/MM specification. SQL-MM 3: 8.2.5

Examples

```
--If you have a regular polygon
SELECT gid, field1, field2, ST_NumInteriorRings(the_geom) AS numholes
FROM sometable;

--If you have multipolygons
--And you want to know the total number of interior rings in the MULTIPOLYGON
SELECT gid, field1, field2, SUM(ST_NumInteriorRings(the_geom)) AS numholes
FROM (SELECT gid, field1, field2, (ST_Dump(the_geom)).geom As the_geom
  FROM sometable) As foo
GROUP BY gid, field1,field2;
```

See Also

ST_NumInteriorRing

7.4.23 ST_NumInteriorRing

ST_NumInteriorRing — Return the number of interior rings of the first polygon in the geometry. Synonym to ST_NumInteriorRings.

Synopsis

integer **ST_NumInteriorRing**(geometry a_polygon);

Description

Return the number of interior rings of the first polygon in the geometry. Synonym to ST_NumInteriorRings. The OpenGIS specs are ambiguous about the exact function naming, so we provide both spellings.

✓ This method implements the SQL/MM specification. SQL-MM 3: 8.2.5

See Also

ST_NumInteriorRings

7.4.24 ST_NumPoints

ST_NumPoints — Return the number of points in an ST_LineString or ST_CircularString value.

Synopsis

integer **ST_NumPoints**(geometry g1);

Description

Return the number of points in an ST_LineString or ST_CircularString value. Prior to 1.4 only works with Linestrings as the specs state. From 1.4 forward this is an alias for ST_NPoints which returns number of vertexes for not just line strings. Consider using ST_NPoints instead which is multi-purpose and works with many geometry types.

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

✓ This method implements the SQL/MM specification. SQL-MM 3: 7.2.4

Examples

```
SELECT ST_NumPoints(ST_GeomFromText('LINESTRING(77.29 29.07,77.42 29.26,77.27 29.31,77.29 ↩
    29.07)'));
    --result
    4
```

See Also

ST_NPoints

7.4.25 ST_PointN

ST_PointN — Return the Nth point in the first linestring or circular linestring in the geometry. Return NULL if there is no linestring in the geometry.

Synopsis

geometry **ST_PointN**(geometry a_linestring, integer n);

Description

Return the Nth point in the first linestring or circular linestring in the geometry. Return NULL if there is no linestring in the geometry.

> **Note**
> Index is 1-based as for OGC specs since version 0.8.0. Previous versions implemented this as 0-based instead.

> **Note**
> If you want to get the nth point of each line string in a multilinestring, use in conjunction with ST_Dump

✓ This method implements the OpenGIS 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

Examples

```
-- 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)
```

See Also

ST_NPoints

7.4.26 ST_SRID

ST_SRID — Returns the spatial reference identifier for the ST_Geometry as defined in spatial_ref_sys table.

Synopsis

integer **ST_SRID**(geometry g1);

Description

Returns the spatial reference identifier for the ST_Geometry as defined in Section 4.3.1 table.

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 OpenGIS 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 4.3.1,ST_GeomFromText, ST_SetSRID, ST_Transform

7.4.27 ST_StartPoint

ST_StartPoint — Returns the first point of a `LINESTRING` geometry as a `POINT`.

Synopsis

geometry **ST_StartPoint**(geometry geomA);

Description

Returns the first point of a `LINESTRING` geometry as a `POINT` or `NULL` if the input parameter is not a `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.

Examples

```
SELECT ST_AsText(ST_StartPoint(’LINESTRING(0 1, 0 2)’::geometry));
 st_astext
------------
 POINT(0 1)
(1 row)

SELECT ST_StartPoint(’POINT(0 1)’::geometry) IS NULL AS is_null;
  is_null
----------
 t
(1 row)

--3d line
SELECT ST_AsEWKT(ST_StartPoint(’LINESTRING(0 1 1, 0 2 2)’::geometry));
 st_asewkt
------------
 POINT(0 1 1)
(1 row)
```

See Also

ST_EndPoint, ST_PointN

7.4.28 ST_Summary

ST_Summary — Returns a text summary of the contents of the `ST_Geometry`.

Synopsis

text **ST_Summary**(geometry g);

Description

Returns a text summary of the contents of the geometry.

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

Examples

```
SELECT ST_Summary(ST_GeomFromText('LINESTRING(0 0, 1 1)')) As good_line,
  ST_Summary(ST_GeomFromText('POLYGON((0 0, 1 1, 1 2, 1 1, 0 0))')) As bad_poly
--results
    good_line       |        bad_poly
----------------------+-------------------------
            |
Line[B] with 2 points : Polygon[B] with 1 rings
            :    ring 0 has 5 points
            :

--3d polygon
SELECT ST_Summary(ST_GeomFromEWKT('LINESTRING(0 0 1, 1 1 1)')) As good_line,
  ST_Summary(ST_GeomFromEWKT('POLYGON((0 0 1, 1 1 2, 1 2 3, 1 1 1, 0 0 1))')) As poly

--results
    good_line       |        poly
----------------------+-------------------------
            |
Line[ZB] with 2 points : Polygon[ZB] with 1 rings
            :    ring 0 has 5 points
            :
```

See Also

ST_IsValid, ST_IsValidReason

7.4.29 ST_X

ST_X — Return the X coordinate of the point, or NULL if not available. Input must be a point.

Synopsis

float **ST_X**(geometry a_point);

Description

Return the X coordinate of the point, or NULL if not available. Input must be a point.

Note
If you want to get the max min x values of any geometry look at ST_XMin, ST_XMax functions.

✓ This method implements the SQL/MM specification. SQL-MM 3: 6.1.3

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

Examples

```
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)
```

See Also

ST_Centroid, ST_GeomFromEWKT, ST_M, ST_XMax, ST_XMin, ST_Y, ST_Z

7.4.30 ST_Y

ST_Y — Return the Y coordinate of the point, or NULL if not available. Input must be a point.

Synopsis

float **ST_Y**(geometry a_point);

Description

Return the Y coordinate of the point, or NULL if not available. Input must be a point.

✓ This method implements the OpenGIS 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.

Examples

```
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)
```

See Also

ST_Centroid, ST_GeomFromEWKT, ST_M, ST_X, ST_YMax, ST_YMin, ST_Z

7.4.31 ST_Z

ST_Z — Return the Z coordinate of the point, or NULL if not available. Input must be a point.

Synopsis

float **ST_Z**(geometry a_point);

Description

Return the Z coordinate of the point, or NULL if not available. Input must be a point.

✓ This method implements the SQL/MM specification.

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

Examples

```
SELECT ST_Z(ST_GeomFromEWKT('POINT(1 2 3 4)'));
 st_z
------
  3
(1 row)
```

See Also

ST_GeomFromEWKT, ST_M, ST_X, ST_Y, ST_ZMax, ST_ZMin

7.4.32 ST_Zmflag

ST_Zmflag — Returns ZM (dimension semantic) flag of the geometries as a small int. Values are: 0=2d, 1=3dm, 2=3dz, 3=4d.

Synopsis

smallint **ST_Zmflag**(geometry geomA);

Description

Returns ZM (dimension semantic) flag of the geometries as a small int. Values are: 0=2d, 1=3dm, 2=3dz, 3=4d.

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

✓ This method supports Circular Strings and Curves

Examples

```
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
```

See Also

ST_CoordDim, ST_NDims, ST_Dimension

7.5 Geometry Editors

7.5.1 ST_AddPoint

ST_AddPoint — Adds a point to a LineString before point <position> (0-based index).

Synopsis

geometry **ST_AddPoint**(geometry linestring, geometry point);

geometry **ST_AddPoint**(geometry linestring, geometry point, integer position);

Description

Adds a point to a LineString before point <position> (0-based index). Third parameter can be omitted or set to -1 for appending.

Availability: 1.1.0

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

Examples

```
--guarantee all linestrings in a table are closed
--by adding the start point of each linestring to the end of the line string
--only for those that are not closed
UPDATE sometable
SET the_geom = ST_AddPoint(the_geom, ST_StartPoint(the_geom))
FROM sometable
WHERE ST_IsClosed(the_geom) = false;

--Adding point to a 3-d line
SELECT ST_AsEWKT(ST_AddPoint(ST_GeomFromEWKT('LINESTRING(0 0 1, 1 1 1)'), ST_MakePoint ↩
    (1, 2, 3)));

--result
st_asewkt
----------
LINESTRING(0 0 1,1 1 1,1 2 3)
```

See Also

ST_RemovePoint, ST_SetPoint

7.5.2 ST_Affine

ST_Affine — Applies a 3d affine transformation to the geometry to do things like translate, rotate, scale in one step.

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.

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 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_RotateZ();
 SELECT ST_AsEWKT(ST_Affine(the_geom,  cos(pi()), -sin(pi()), 0,  sin(pi()), cos(pi()), 0,  ↩
      0, 0, 1,  0, 0, 0)) As using_affine,
   ST_AsEWKT(ST_RotateZ(the_geom, pi())) As using_rotatez
  FROM (SELECT ST_GeomFromEWKT('LINESTRING(1 2 3, 1 4 3)') As the_geom) As foo;
    using_affine         |        using_rotatez
-----------------------------+-----------------------------
 LINESTRING(-1 -2 3,-1 -4 3) | LINESTRING(-1 -2 3,-1 -4 3)
(1 row)

--Rotate a 3d line 180 degrees in both the x and z axis
SELECT ST_AsEWKT(ST_Affine(the_geom, cos(pi()), -sin(pi()), 0, sin(pi()), cos(pi()), -sin( ↩
    pi()), 0, sin(pi()), cos(pi()), 0, 0, 0))
  FROM (SELECT ST_GeomFromEWKT('LINESTRING(1 2 3, 1 4 3)') As the_geom) As foo;
       st_asewkt
-------------------------------
 LINESTRING(-1 -2 -3,-1 -4 -3)
(1 row)
```

See Also

ST_Rotate, ST_Scale, ST_Translate, ST_TransScale

7.5.3 ST_Force_2D

ST_Force_2D — Forces the geometries into a "2-dimensional mode" so that all output representations will only have the X and Y coordinates.

Synopsis

geometry **ST_Force_2D**(geometry geomA);

Description

Forces the geometries into a "2-dimensional mode" so that all output representations will only have the X and Y coordinates. This is useful for force OGC-compliant output (since OGC only specifies 2-D geometries).

✓ This method supports Circular Strings and Curves

Examples

```
SELECT ST_AsEWKT(ST_Force_2D(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_Force_2D('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))
```

See Also

ST_Force_3D

7.5.4 ST_Force_3D

ST_Force_3D — Forces the geometries into XYZ mode. This is an alias for ST_Force_3DZ.

Synopsis

geometry **ST_Force_3D**(geometry geomA);

Description

Forces the geometries into XYZ mode. This is an alias for ST_Force_3DZ. If a geometry has no Z component, then a 0 Z coordinate is tacked on.

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

✓ This method supports Circular Strings and Curves

Examples

```
    --Nothing happens to an already 3D geometry
    SELECT ST_AsEWKT(ST_Force_3D(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_Force_3D('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))
```

See Also

ST_AsEWKT, ST_Force_2D, ST_Force_3DM, ST_Force_3DZ

7.5.5 ST_Force_3DZ

ST_Force_3DZ — Forces the geometries into XYZ mode. This is a synonym for ST_Force_3D.

Synopsis

geometry **ST_Force_3DZ**(geometry geomA);

Description

Forces the geometries into XYZ mode. This is a synonym for ST_Force_3DZ. If a geometry has no Z component, then a 0 Z coordinate is tacked on.

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

✓ This method supports Circular Strings and Curves

Examples

```
--Nothing happens to an already 3D geometry
SELECT ST_AsEWKT(ST_Force_3DZ(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_Force_3DZ('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))
```

See Also

ST_AsEWKT, ST_Force_2D, ST_Force_3DM, ST_Force_3D

7.5.6 ST_Force_3DM

ST_Force_3DM — Forces the geometries into XYM mode.

Synopsis

geometry **ST_Force_3DM**(geometry geomA);

Description

Forces the geometries into XYM mode. If a geometry has no M component, then a 0 M coordinate is tacked on. If it has a Z component, then Z is removed

✓ This method supports Circular Strings and Curves

Examples

```
--Nothing happens to an already 3D geometry
SELECT ST_AsEWKT(ST_Force_3DM(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_Force_3DM('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))
```

See Also

ST_AsEWKT, ST_Force_2D, ST_Force_3DM, ST_Force_3D, ST_GeomFromEWKT

7.5.7 ST_Force_4D

ST_Force_4D — Forces the geometries into XYZM mode.

Synopsis

geometry **ST_Force_4D**(geometry geomA);

Description

Forces the geometries into XYZM mode. 0 is tacked on for missing Z and M dimensions.

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

✓ This method supports Circular Strings and Curves

Examples

```
--Nothing happens to an already 3D geometry
SELECT ST_AsEWKT(ST_Force_4D(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_Force_4D('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))
```

See Also

ST_AsEWKT, ST_Force_2D, ST_Force_3DM, ST_Force_3D

7.5.8 ST_Force_Collection

ST_Force_Collection — Converts the geometry into a GEOMETRYCOLLECTION.

Synopsis

geometry **ST_Force_Collection**(geometry geomA);

Description

Converts the geometry into a GEOMETRYCOLLECTION. This is useful for simplifying the WKB representation.

Availability: 1.2.2, prior to 1.3.4 this function will crash with 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

Examples

```
SELECT  ST_AsEWKT(ST_Force_Collection('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_Force_Collection('CIRCULARSTRING(220227 150406,2220227 150407,220227  ↩
      150406)'));
                   st_astext
--------------------------------------------------------------------------------
 GEOMETRYCOLLECTION(CIRCULARSTRING(220227 150406,2220227 150407,220227 150406))
(1 row)
```

See Also

ST_AsEWKT, ST_Force_2D, ST_Force_3DM, ST_Force_3D, ST_GeomFromEWKT

7.5.9 ST_ForceRHR

ST_ForceRHR — Forces the orientation of the vertices in a polygon to follow the Right-Hand-Rule.

Synopsis

boolean **ST_ForceRHR**(geometry g);

Description

Forces the orientation of the vertices in a polygon to follow the Right-Hand-Rule. In GIS terminology, this means that 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 supports 3d and will not drop the z-index.

Examples

```
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)
```

See Also

ST_BuildArea, ST_Polygonize, ST_Reverse

7.5.10 ST_LineMerge

ST_LineMerge — Returns a (set of) LineString(s) formed by sewing together a MULTILINESTRING.

Synopsis

geometry **ST_LineMerge**(geometry amultilinestring);

Description

Returns a (set of) LineString(s) formed by sewing together the constituent line work of a MULTILINESTRING.

Note
Only use with MULTILINESTRING/LINESTRINGs. If you feed a polygon or geometry collection into this function, it will return an empty GEOMETRYCOLLECTION

Availability: 1.1.0

Note
requires GEOS >= 2.1.0

Examples

```
SELECT ST_AsText(ST_LineMerge(
ST_GeomFromText('MULTILINESTRING((-29 -27,-30 -29.7,-36 -31,-45 -33),(-46 -32))')
    )
);
st_astext
-------------------------------------------------------------------------------------------------- ←

LINESTRING(-29 -27,-30 -29.7,-36 -31,-45 -33,-46 -32)
(1 row)
```

```
--If can't be merged - original MULTILINESTRING is returned
SELECT ST_AsText(ST_LineMerge(
ST_GeomFromText('MULTILINESTRING((-29 -27,-30 -29.7,-36 -31,-45 -33),(-45.2 -33.2,-46 -32)) ↩
    ')
)
);
st_astext
----------------
MULTILINESTRING((-45.2 -33.2,-46 -32),(-29 -27,-30 -29.7,-36 -31,-45 -33))
```

See Also

ST_Segmentize, ST_Line_Substring

7.5.11 ST_CollectionExtract

ST_CollectionExtract — Given a GEOMETRYCOLLECTION, returns a MULTI* geometry consisting only of the specified type. Sub-geometries that are not the specified type are ignored. If there are no sub-geometries of the right type, an EMPTY collection will be returned. Only points, lines and polygons are supported. Type numbers are 1 == POINT, 2 == LINESTRING, 3 == POLYGON.

Synopsis

geometry **ST_CollectionExtract**(geometry collection, integer type);

Description

Given a GEOMETRYCOLLECTION, returns a MULTI* geometry consisting only of the specified type. Sub-geometries that are not the specified type are ignored. If there are no sub-geometries of the right type, an EMPTY collection will be returned. Only points, lines and polygons are supported. Type numbers are 1 == POINT, 2 == LINESTRING, 3 == POLYGON.

Availability: 1.5.0

Examples

```
-- Constants: 1 == POINT, 2 == LINESTRING, 3 == POLYGON
SELECT ST_AsText(ST_CollectionExtract(ST_GeomFromText('GEOMETRYCOLLECTION( ↩
    GEOMETRYCOLLECTION(POINT(0 0)))'),1));
st_astext
---------------
MULTIPOINT(0 0)
(1 row)

SELECT ST_AsText(ST_CollectionExtract(ST_GeomFromText('GEOMETRYCOLLECTION( ↩
    GEOMETRYCOLLECTION(LINESTRING(0 0, 1 1)),LINESTRING(2 2, 3 3))'),2));
st_astext
---------------
MULTILINESTRING((0 0, 1 1), (2 2, 3 3))
(1 row)
```

See Also

ST_Multi

7.5.12 ST_Multi

ST_Multi — Returns the geometry as a MULTI* geometry. If the geometry is already a MULTI*, it is returned unchanged.

Synopsis

geometry **ST_Multi**(geometry g1);

Description

Returns the geometry as a MULTI* geometry. If the geometry is already a MULTI*, it is returned unchanged.

Examples

```
SELECT ST_AsText(ST_Multi(ST_GeomFromText('POLYGON((743238 2967416,743238 2967450,
      743265 2967450,743265.625 2967416,743238 2967416))')));
      st_astext
      --------------------------------------------------------------------------------------------

      MULTIPOLYGON(((743238 2967416,743238 2967450,743265 2967450,743265.625 2967416,
      743238 2967416)))
      (1 row)
```

See Also

ST_AsText

7.5.13 ST_RemovePoint

ST_RemovePoint — Removes point from a linestring. Offset is 0-based.

Synopsis

geometry **ST_RemovePoint**(geometry linestring, integer offset);

Description

Removes point from a linestring. Useful for turning a closed ring into an open line string

Availability: 1.1.0

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

Examples

```
--guarantee no LINESTRINGS are closed
--by removing the end point.  The below assumes the_geom is of type LINESTRING
UPDATE sometable
  SET the_geom = ST_RemovePoint(the_geom, ST_NPoints(the_geom) - 1)
  FROM sometable
  WHERE ST_IsClosed(the_geom) = true;
```

See Also

ST_AddPoint, ST_NPoints, ST_NumPoints

7.5.14 ST_Reverse

ST_Reverse — Returns the geometry with vertex order reversed.

Synopsis

geometry **ST_Reverse**(geometry g1);

Description

Can be used on any geometry and reverses the order of the vertexes.

Examples

```
SELECT ST_AsText(the_geom) as line, ST_AsText(ST_Reverse(the_geom)) As reverseline
FROM
(SELECT ST_MakeLine(ST_MakePoint(1,2),
    ST_MakePoint(1,10)) As the_geom) as foo;
--result
    line         |     reverseline
---------------------+----------------------
LINESTRING(1 2,1 10) | LINESTRING(1 10,1 2)
```

7.5.15 ST_Rotate

ST_Rotate — This is a synonym for ST_RotateZ

Synopsis

geometry **ST_Rotate**(geometry geomA, float rotZRadians);

Description

This is a synonym for ST_RotateZ.. Rotates geometry rotZRadians about the Z-axis.

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

Examples

See Also

ST_Affine, ST_RotateX, ST_RotateY, ST_RotateZ

7.5.16 ST_RotateX

ST_RotateX — Rotate a geometry rotRadians about the X axis.

Synopsis

geometry **ST_RotateX**(geometry geomA, float rotRadians);

Description

Rotate 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)`.

Availability: 1.1.2. Name changed from RotateX to ST_RotateX in 1.2.2

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

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

7.5.17 ST_RotateY

ST_RotateY — Rotate a geometry rotRadians about the Y axis.

Synopsis

geometry **ST_RotateY**(geometry geomA, float rotRadians);

Description

Rotate 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

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

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, Rotate around Point, Create Ellipse functions

7.5.18 ST_RotateZ

ST_RotateZ — Rotate a geometry rotRadians about the Z axis.

Synopsis

geometry **ST_RotateZ**(geometry geomA, float rotRadians);

Description

Rotate a geometry geomA - rotRadians about the Z axis.

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).`

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

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(the_geom, pi()/2))
FROM (SELECT ST_LineToCurve(ST_Buffer(ST_GeomFromText('POINT(234 567)'), 3)) As the_geom)  ↩
    As foo;

                             st_asewkt
----------------------------------------------------------------------------------------------------

 CURVEPOLYGON(CIRCULARSTRING(-567 237,-564.87867965644 236.12132034356,-564  ↩
     234,-569.12132034356 231.87867965644,-567 237))
```

See Also

ST_Affine, ST_RotateX, ST_RotateY, Rotate around Point, Create Ellipse functions

7.5.19 ST_Scale

ST_Scale — Scales the geometry to a new size by multiplying the ordinates with the parameters. Ie: ST_Scale(geom, Xfactor, Yfactor, Zfactor).

Synopsis

geometry **ST_Scale**(geometry geomA, float XFactor, float YFactor, float ZFactor);
geometry **ST_Scale**(geometry geomA, float XFactor, float YFactor);

Description

Scales the geometry to a new size by multiplying the ordinates with the parameters. Ie: ST_Scale(geom, Xfactor, Yfactor, Zfactor).

Note
`ST_Scale(geomA, XFactor, YFactor, ZFactor)` is short-hand for `ST_Affine(geomA, XFactor, 0, 0, 0, YFactor, 0, 0, 0, ZFactor, 0, 0, 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

```
--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)
```

See Also

ST_Affine, ST_TransScale

7.5.20 ST_Segmentize

ST_Segmentize — Return a modified geometry having no segment longer than the given distance. Distance computation is performed in 2d only.

Synopsis

geometry **ST_Segmentize**(geometry geomA, float max_length);

Description

Returns a modified geometry having no segment longer than the given distance. Distance computation is performed in 2d only.

Availability: 1.2.2

Note
This will only increase segments. It will not lengthen segments shorter than max length

Examples

```
SELECT ST_AsText(ST_Segmentize(
ST_GeomFromText('MULTILINESTRING((-29 -27,-30 -29.7,-36 -31,-45 -33),(-45 -33,-46 -32))')
    ,5)
);
st_astext
-------------------------------------------------------------------------------------------------

MULTILINESTRING((-29 -27,-30 -29.7,-34.886615700134 -30.758766735029,-36 -31,
-40.8809353009198 -32.0846522890933,-45 -33),
(-45 -33,-46 -32))
(1 row)
```

```
SELECT ST_AsText(ST_Segmentize(ST_GeomFromText('POLYGON((-29 28, -30 40, -29 28))'),10));
st_astext
-----------------------
POLYGON((-29 28,-29.8304547985374 37.9654575824488,-30 40,-29.1695452014626  ↩
    30.0345424175512,-29 28))
(1 row)
```

See Also

ST_Line_Substring

7.5.21 ST_SetPoint

ST_SetPoint — Replace point N of linestring with given point. Index is 0-based.

Synopsis

geometry **ST_SetPoint**(geometry linestring, integer zerobasedposition, geometry point);

Description

Replace point N of linestring with given point. Index is 0-based. This is especially useful in triggers when trying to maintain relationship of joints when one vertex moves.

Availability: 1.1.0

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

Examples

```
--Change first point in line string from -1 3 to -1 1
SELECT ST_AsText(ST_SetPoint('LINESTRING(-1 2,-1 3)', 0, 'POINT(-1 1)'));
     st_astext
-----------------------
 LINESTRING(-1 1,-1 3)

---Change last point in a line string (lets play with 3d linestring this time)
SELECT ST_AsEWKT(ST_SetPoint(foo.the_geom, ST_NumPoints(foo.the_geom) - 1, ST_GeomFromEWKT ↩
    ('POINT(-1 1 3)')))
FROM (SELECT ST_GeomFromEWKT('LINESTRING(-1 2 3,-1 3 4, 5 6 7)') As the_geom) As foo;
     st_asewkt
-----------------------
LINESTRING(-1 2 3,-1 3 4,-1 1 3)
```

See Also

ST_AddPoint,ST_NPoints, ST_NumPoints, ST_PointN, ST_RemovePoint

7.5.22 ST_SetSRID

ST_SetSRID — Sets the SRID on a geometry to a particular integer value.

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 OpenGIS 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 4.3.1, ST_AsEWKT, ST_Point, ST_SRID,ST_Transform, UpdateGeometrySRID

7.5.23 ST_SnapToGrid

ST_SnapToGrid — Snap all points of the input geometry to the grid defined by its origin and cell size. Remove consecutive points falling on the same cell, eventually returning NULL if output points are not enough to define a geometry of the given type. Collapsed geometries in a collection are stripped from it. Useful for reducing precision.

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);

Description

Variant 1,2,3: Snap all points of the input geometry to the grid defined by its origin and cell size. Remove consecutive points falling on the same cell, eventually returning NULL if output points are not enough to define a geometry of the given type. Collapsed geometries in a collection are stripped from it.

Variant 4: Introduced 1.1.0 - Snap all points of the input geometry to the grid defined by its origin (the second argument, must be a point) and cell sizes. Specify 0 as size for any dimension you don't want to snap to a grid.

Note
The returned geometry might loose its simplicity (see ST_IsSimple).

Note
Before release 1.1.0 this function always returned a 2d geometry. Starting at 1.1.0 the returned geometry will have same dimensionality as the input one with higher dimension values untouched. Use the version taking a second geometry argument to define all grid dimensions.

Availability: 1.0.0RC1

Availability: 1.1.0 - Z and M support

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

Examples

```
--Snap your geometries to a precision grid of 10^-3
UPDATE mytable
   SET the_geom = ST_SnapToGrid(the_geom, 0.001);

SELECT ST_AsText(ST_SnapToGrid(
      ST_GeomFromText('LINESTRING(1.1115678 2.123, 4.111111 3.2374897, 4.11112 3.23748667) ↩
          '),
      0.001)
    );
        st_astext
-------------------------------------
 LINESTRING(1.112 2.123,4.111 3.237)
 --Snap a 4d geometry
SELECT ST_AsEWKT(ST_SnapToGrid(
  ST_GeomFromEWKT('LINESTRING(-1.1115678 2.123 2.3456 1.11111,
    4.111111 3.2374897 3.1234 1.1111, -1.11111112 2.123 2.3456 1.1111112)'),
 ST_GeomFromEWKT('POINT(1.12 2.22 3.2 4.4444)'),
 0.1, 0.1, 0.1, 0.01) );
                  st_asewkt
------------------------------------------------------------------------------
 LINESTRING(-1.08 2.12 2.3 1.1144,4.12 3.22 3.1 1.1144,-1.08 2.12 2.3 1.1144)


--With a 4d geometry - the ST_SnapToGrid(geom,size) only touches x and y coords but keeps m ↩
     and z the same
SELECT ST_AsEWKT(ST_SnapToGrid(ST_GeomFromEWKT('LINESTRING(-1.1115678 2.123 3 2.3456,
    4.111111 3.2374897 3.1234 1.1111)'),
     0.01)      );
            st_asewkt
```

```
---------------------------------------------------------
 LINESTRING(-1.11 2.12 3 2.3456,4.11 3.24 3.1234 1.1111)
```

See Also

ST_AsEWKT, ST_AsText, ST_GeomFromText, ST_GeomFromEWKT, ST_Simplify

7.5.24 ST_Transform

ST_Transform — Returns a new geometry with its coordinates transformed to the SRID referenced by the integer parameter.

Synopsis

geometry **ST_Transform**(geometry g1, integer srid);

Description

Returns a new geometry with its coordinates transformed to spatial reference system referenced by the SRID integer parameter. The destination SRID must exist in the `SPATIAL_REF_SYS` table.

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

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+

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

✓ This method implements the SQL/MM specification. SQL-MM 3: 5.1.6

✓ This method supports Circular Strings and Curves

Examples

Change Mass state plane US feet geometry to WGS 84 long lat

```
SELECT ST_AsText(ST_Transform(ST_GeomFromText('POLYGON((743238 2967416,743238 2967450,
  743265 2967450,743265.625 2967416,743238 2967416))',2249),4326)) As wgs_geom;

 wgs_geom
---------------------------
 POLYGON((-71.1776848522251 42.3902896512902,-71.1776843766326 42.3903829478009,
-71.1775844305465 42.3903826677917,-71.1775825927231 42.3902893647987,-71.177684
8522251 42.3902896512902));
(1 row)

--3D Circular String example
SELECT ST_AsEWKT(ST_Transform(ST_GeomFromEWKT('SRID=2249;CIRCULARSTRING(743238 2967416  ↩
    1,743238 2967450 2,743265 2967450 3,743265.625 2967416 3,743238 2967416 4)'),4326));

        st_asewkt
--------------------------------------------------------------------------------------
 SRID=4326;CIRCULARSTRING(-71.1776848522251 42.3902896512902 1,-71.1776843766326  ↩
     42.3903829478009 2,
 -71.1775844305465 42.3903826677917 3,
 -71.1775825927231 42.3902893647987 3,-71.1776848522251 42.3902896512902 4)
```

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_the_geom_26986_parcels
  ON parcels
  USING gist
  (ST_Transform(the_geom, 26986))
  WHERE the_geom IS NOT NULL;
```

Configuring transformation behaviour

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 behaviour can be configured on a per-SRID basis by 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

PostGIS_Full_Version, ST_AsText, ST_SetSRID, UpdateGeometrySRID

7.5.25 ST_Translate

ST_Translate — Translates the geometry to a new location using the numeric parameters as offsets. Ie: ST_Translate(geom, X, Y) or ST_Translate(geom, X, Y,Z).

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));
                             st_astext
----------------------------------------------------------------------------------------------------

 GEOMETRYCOLLECTION(CURVEPOLYGON(CIRCULARSTRING(5 5,4.12 2.878,2 2,-0.121 7.1213,7 9,9 11,5 ↩
      5)),POINT(2 5))
```

See Also

ST_Affine, ST_AsText, ST_GeomFromText

7.5.26 ST_TransScale

ST_TransScale — Translates the geometry using the deltaX and deltaY args, then scales it using the XFactor, YFactor args, working in 2D only.

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));
                              st_astext
----------------------------------------------------------------------------------------------------

 CURVEPOLYGON(CIRCULARSTRING(714 2276,711.363961030679 2267.51471862576,705 2264,698.636038969321 2284.48528137424,714 2276))
```

See Also

ST_Affine, ST_Translate

7.6 Geometry Outputs

7.6.1 ST_AsBinary

ST_AsBinary — Return the Well-Known Binary (WKB) representation of the geometry/geography without SRID meta data.

Synopsis

bytea **ST_AsBinary**(geometry g1);
bytea **ST_AsBinary**(geography g1);
bytea **ST_AsBinary**(geometry g1, text NDR_or_XDR);

Description

Returns the Well-Known Binary representation of the geometry. There are 2 variants of the function. The first variant takes no endian encoding paramater and defaults to little endian. The second variant takes a second argument denoting the encoding - using little-endian ('NDR') or big-endian ('XDR') encoding.

This is useful in binary cursors to pull data out of the database without converting it to a string representation.

Note
The WKB spec does not include the SRID. To get the OGC WKB with SRID format use ST_AsEWKB

Note
ST_AsBinary is the reverse of ST_GeomFromWKB for geometry. Use ST_GeomFromWKB to convert to a postgis geometry from ST_AsBinary representation.

Availability: 1.5.0 geography support was introduced.

✓ This method implements the OpenGIS 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

Examples

```
SELECT ST_AsBinary(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))',4326));

       st_asbinary
--------------------------------
\001\003\000\000\000\001\000\000\000\005
\000\000\000\000\000\000\000\000\000\000
\000\000\000\000\000\000\000\000\000\000
\000\000\000\000\000\000\000\000\000\000
\000\000\000\360?\000\000\000\000\000\000
\360?\000\000\000\000\000\000\360?\000\000
\000\000\000\000\360?\000\000\000\000\000
\000\000\000\000\000\000\000\000\000\000\000
\000\000\000\000\000\000\000\000
(1 row)
```

```
SELECT ST_AsBinary(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))',4326), 'XDR');
       st_asbinary
--------------------------------
\000\000\000\000\003\000\000\000\001\000\000\000\005\000\000\000\000\000
\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000
\000?\360\000\000\000\000\000\000?\360\000\000\000\000\000\000?\360\000\000
\000\000\000\000?\360\000\000\000\000\000\000\000\000\000\000\000\000\000\000
\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000
(1 row)
```

See Also

ST_AsEWKB, ST_AsEWKT, ST_AsText, ST_GeomFromEWKB

7.6.2 ST_AsEWKB

ST_AsEWKB — Return the Well-Known Binary (WKB) representation of the geometry with SRID meta data.

Synopsis

bytea **ST_AsEWKB**(geometry g1);
bytea **ST_AsEWKB**(geometry g1, text NDR_or_XDR);

Description

Returns the Well-Known Binary representation of the geometry with SRID metadata. There are 2 variants of the function. The first variant takes no endian encoding paramater and defaults to little endian. The second variant takes a second argument denoting the encoding - using little-endian ('NDR') or big-endian ('XDR') encoding.

This is useful in binary cursors to pull data out of the database without converting it to a string representation.

Note
The WKB spec does not include the SRID. To get the OGC WKB format use ST_AsBinary

Note
ST_AsEWKB is the reverse of ST_GeomFromEWKB. Use ST_GeomFromEWKB to convert to a postgis geometry from ST_AsEWKB representation.

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

✓ This method supports Circular Strings and Curves

Examples

```
SELECT ST_AsEWKB(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))',4326));

       st_asewkb
--------------------------------
\001\003\000\000 \346\020\000\000\001\000
\000\000\005\000\000\000\000
\000\000\000\000\000\000\000\000
\000\000\000\000\000\000\000\000\000
\000\000\000\000\000\000\000\000\000\000
\000\000\360?\000\000\000\000\000\000\360?
\000\000\000\000\000\000\360?\000\000\000\000\000
\000\360?\000\000\000\000\000\000\000\000\000\000\000
\000\000\000\000\000\000\000\000\000\000\000\000\000
(1 row)
```

```
      SELECT ST_AsEWKB(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))',4326), 'XDR');
       st_asewkb
--------------------------------
\000 \000\000\003\000\000\020\346\000\000\000\001\000\000\000\005\000\000\000\000\
000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000?
\360\000\000\000\000\000\000?\360\000\000\000\000\000\000?\360\000\000\000\000
\000\000?\360\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000
\000\000\000\000\000\000\000\000\000\000\000\000\000
```

See Also

ST_AsBinary, ST_AsEWKT, ST_AsText, ST_GeomFromEWKT, ST_SRID

7.6.3 ST_AsEWKT

ST_AsEWKT — Return the Well-Known Text (WKT) representation of the geometry with SRID meta data.

Synopsis

text **ST_AsEWKT**(geometry g1);

Description

Returns the Well-Known Text representation of the geometry prefixed with the SRID.

Note
The WKT spec does not include the SRID. To get the OGC WKT format use ST_AsText

WKT format does not maintain precision so to prevent floating truncation, use ST_AsBinary or ST_AsEWKB format for transport.

Note
ST_AsEWKT is the reverse of ST_GeomFromEWKT. Use ST_GeomFromEWKT to convert to a postgis geometry from ST_AsEWKT representation.

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

✓ This method supports Circular Strings and Curves

Examples

```
SELECT ST_AsEWKT('0103000020E61000000100000005000000000000
      000000000000000000000000000000000000000000000000000000
      F03F000000000000F03F000000000000F03F000000000000F03
      F000000000000000000000000000000000000000000000000'::geometry);

       st_asewkt
--------------------------------
SRID=4326;POLYGON((0 0,0 1,1 1,1 0,0 0))
(1 row)

SELECT ST_AsEWKT('0108000080030000000000000060 ←
    E30A4100000000785C0241000000000000F03F0000000018
E20A4100000000485F024100000000000000400000000018
E20A4100000000305C02410000000000000840')

--st_asewkt---
CIRCULARSTRING(220268 150415 1,220227 150505 2,220227 150406 3)
```

See Also

ST_AsBinaryST_AsEWKBST_AsText, ST_GeomFromEWKT

7.6.4 ST_AsGeoJSON

ST_AsGeoJSON — Return the geometry as a GeoJSON element.

Synopsis

text **ST_AsGeoJSON**(geometry g1);
text **ST_AsGeoJSON**(geography g1);
text **ST_AsGeoJSON**(geometry g1, integer max_decimal_digits);

text **ST_AsGeoJSON**(geography g1, integer max_decimal_digits);
text **ST_AsGeoJSON**(geometry g1, integer max_decimal_digits, integer options);
text **ST_AsGeoJSON**(geography g1, integer max_decimal_digits, integer options);
text **ST_AsGeoJSON**(integer gj_version, geometry g1);
text **ST_AsGeoJSON**(integer gj_version, geography g1);
text **ST_AsGeoJSON**(integer gj_version, geometry g1, integer max_decimal_digits);
text **ST_AsGeoJSON**(integer gj_version, geography g1, integer max_decimal_digits);
text **ST_AsGeoJSON**(integer gj_version, geometry g1, integer max_decimal_digits, integer options);
text **ST_AsGeoJSON**(integer gj_version, geography g1, integer max_decimal_digits, integer options);

Description

Return the geometry as a Geometry Javascript Object Notation (GeoJSON) element. (Cf GeoJSON specifications 1.0). 2D and 3D Geometries are both supported. GeoJSON only support SFS 1.1 geometry type (no curve support for example).

The gj_version parameter is the major version of the GeoJSON spec. If specified, must be 1.

The third argument may be used to reduce the maximum number of decimal places used in output (defaults to 15).

The last ’options’ argument could be used to add Bbox or Crs in GeoJSON output:

- 0: means no option (default value)
- 1: GeoJSON Bbox
- 2: GeoJSON Short CRS (e.g EPSG:4326)
- 4: GeoJSON Long CRS (e.g urn:ogc:def:crs:EPSG::4326)

Version 1: ST_AsGeoJSON(geom) / precision=15 version=1 options=0

Version 2: ST_AsGeoJSON(geom, precision) / version=1 options=0

Version 3: ST_AsGeoJSON(geom, precision, options) / version=1

Version 4: ST_AsGeoJSON(version, geom) / precision=15 options=0

Version 5: ST_AsGeoJSON(version, geom, precision) /options=0

Version 6: ST_AsGeoJSON(version, geom, precision,options)

Availability: 1.3.4

Availability: 1.5.0 geography support was introduced.

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

Examples

GeoJSON format is generally more efficient than other formats for use in ajax mapping. One popular javascript client that supports this is Open Layers. Example of its use is OpenLayers GeoJSON Example

```
SELECT ST_AsGeoJSON(the_geom) from fe_edges limit 1;
             st_asgeojson
----------------------------------------------------------------------------------------------------


{"type":"MultiLineString","coordinates":[[[-89.734634999999997,31.492072000000000],
[-89.734955999999997,31.492237999999997]]]}
(1 row)
--3d point
SELECT ST_AsGeoJSON('LINESTRING(1 2 3, 4 5 6)');
```

```
st_asgeojson
-----------------------------------------------------------------------------------------
 {"type":"LineString","coordinates":[[1,2,3],[4,5,6]]}
```

7.6.5 ST_AsGML

ST_AsGML — Return the geometry as a GML version 2 or 3 element.

Synopsis

text **ST_AsGML**(geometry g1);
text **ST_AsGML**(geography g1);
text **ST_AsGML**(geometry g1, integer precision);
text **ST_AsGML**(geography g1, integer precision);
text **ST_AsGML**(integer version, geometry g1);
text **ST_AsGML**(integer version, geography g1);
text **ST_AsGML**(integer version, geometry g1, integer precision);
text **ST_AsGML**(integer version, geography g1, integer precision);
text **ST_AsGML**(integer version, geometry g1, integer precision, integer options);
text **ST_AsGML**(integer version, geography g1, integer precision, integer options);

Description

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 third argument may be used to reduce the maximum number of decimal places used in output (defaults to 15).

GML 2 refer to 2.1.2 version, GML 3 to 3.1.1 version

The last ’options’ argument is a bitfield. It could be used to define CRS output type in GML output, and to declare data as lat/lon:

- 0: GML Short CRS (e.g EPSG:4326), default value
- 1: GML Long CRS (e.g urn:ogc:def:crs:EPSG::4326)
- 16: Declare that datas are lat/lon (e.g srid=4326). Default is to assume that data are planars. This option is usefull for GML 3.1.1 output only, related to axis order.

Note
Availability: 1.3.2
Availability: 1.5.0 geography support was introduced.

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

Examples

```
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>
```

```
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>
```

See Also

ST_GeomFromGML

7.6.6 ST_AsHEXEWKB

ST_AsHEXEWKB — Returns a Geometry in HEXEWKB format (as text) using either little-endian (NDR) or big-endian (XDR) encoding.

Synopsis

text **ST_AsHEXEWKB**(geometry g1, text NDRorXDR);
text **ST_AsHEXEWKB**(geometry g1);

Description

Returns a Geometry in HEXEWKB format (as text) using either little-endian (NDR) or big-endian (XDR) encoding. If no encoding is specified, then NDR is used.

Note
Availability: 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
    00000000000000000000000000000000
    00000000000000000000000000000000F03F
    000000000000F03F000000000000F03F000000000000F03
    F000000000000000000000000000000000000000000000000
```

7.6.7 ST_AsKML

ST_AsKML — Return the geometry as a KML element. Several variants. Default version=2, default precision=15

Synopsis

text **ST_AsKML**(geometry g1);
text **ST_AsKML**(geography g1);
text **ST_AsKML**(geometry g1, integer precision);
text **ST_AsKML**(geography g1, integer precision);
text **ST_AsKML**(integer version, geometry geom1);
text **ST_AsKML**(integer version, geography geom1);
text **ST_AsKML**(integer version, geometry geom1, integer precision);
text **ST_AsKML**(integer version, geography geom1, integer precision);

Description

Return the geometry as a Keyhole Markup Language (KML) element. There are several variants of this function. maximum number of decimal places used in output (defaults to 15) and version default to 2.

Version 1: ST_AsKML(geom) / version=2 precision=15

Version 2: ST_AsKML(geom, max_sig_digits) / version=2

Version 3: ST_AsKML(version, geom) / precision=15

Version 4: ST_AsKML(version, geom, precision)

Note
Requires PostGIS be compiled with Proj support. Use PostGIS_Full_Version to confirm you have proj support compiled in.

Note
Availability: 1.2.2 - later variants that include version param came in 1.3.2

Note
AsKML output will not work with geometries that do not have an 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>
```

See Also

ST_AsSVG, ST_AsGML

7.6.8 ST_AsSVG

ST_AsSVG — Returns a Geometry in SVG path data given a geometry or geography object.

Synopsis

text **ST_AsSVG**(geometry g1);
text **ST_AsSVG**(geography g1);
text **ST_AsSVG**(geometry g1, integer rel);
text **ST_AsSVG**(geography g1, integer rel);
text **ST_AsSVG**(geometry g1, integer rel, integer maxdecimaldigits);
text **ST_AsSVG**(geography g1, integer rel, integer maxdecimaldigits);

Description

Return the geometry as Scalar Vector Graphics (SVG) path data. Use 1 as second argument to have the path data implemented in terms of relative moves, the default (or 0) uses absolute moves. Third argument may be used to reduce the maximum number of decimal digits used in output (defaults to 15). Point geometries will be rendered as cx/cy when 'rel' arg is 0, x/y when 'rel' is 1. Multipoint geometries are delimited by commas (","), GeometryCollection geometries are delimited by semicolons (";").

Note

Availability: 1.2.2 . Availability: 1.4.0 Changed in PostGIS 1.4.0 to include L command in absolute path to conform to http://www.w3.org/TR/SVG/paths.html#PathDataBNF

Examples

```
SELECT ST_AsSVG(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))',4326));

    st_assvg
    --------
    M 0 0 L 0 -1 1 -1 1 0 Z
```

7.6.9 ST_GeoHash

ST_GeoHash — Return a GeoHash representation (geohash.org) of the geometry.

Synopsis

text **ST_GeoHash**(geometry g1);
text **ST_GeoHash**(geometry g1, integer precision);

Description

Return a GeoHash representation (geohash.org) of the geometry. A GeoHash encodes a 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 also be thought of as a box, that contains the actual point.

The one-parameter variant of ST_GeoHash returns a GeoHash based on the input geometry type. Points return a GeoHash with 20 characters of precision (about enough to hold the full double precision of the input). Other types return a GeoHash with a variable amount of precision, based on the size of the feature. Larger features are represented with less precision, smaller features with more precision. The idea is that the box implied by the GeoHash will always contain the input feature.

The two-parameter variant of ST_GeoHash returns a GeoHash with a requested precision. For non-points, the starting point of the calculation is the center of the bounding box of the geometry.

Availability: 1.4.0

Note
ST_GeoHash will not work with geometries that are not in geographic (lon/lat) coordinates.

✓ This method supports Circular Strings and Curves

Examples

```
SELECT ST_GeoHash(ST_SetSRID(ST_MakePoint(-126,48),4326));

   st_geohash
----------------------
 c0w3hf1s70w3hf1s70w3

SELECT ST_GeoHash(ST_SetSRID(ST_MakePoint(-126,48),4326),5);

 st_geohash
------------
 c0w3h
```

See Also

7.6.10 ST_AsText

ST_AsText — Return the Well-Known Text (WKT) representation of the geometry/geography without SRID metadata.

Synopsis

text **ST_AsText**(geometry g1);
text **ST_AsText**(geography g1);

Description

Returns the Well-Known Text representation of the geometry/geography.

Note
The WKT spec does not include the SRID. To get the SRID as part of the data, use the non-standard PostGIS ST_AsEWKT

WKT format does not maintain precision so to prevent floating truncation, use ST_AsBinary or ST_AsEWKB format for transport.

Note
ST_AsText is the reverse of ST_GeomFromText. Use ST_GeomFromText to convert to a postgis geometry from ST_AsText representation.

Availability: 1.5 - support for geography was introduced.

✓ This method implements the OpenGIS 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('01030000000100000005000000000000000000
000000000000000000000000000000000000000000000000
F03F000000000000F03F000000000000F03F000000000000F03
F000000000000000000000000000000000000000000000000');

       st_astext
--------------------------------
 POLYGON((0 0,0 1,1 1,1 0,0 0))
(1 row)
```

See Also

ST_AsBinary, ST_AsEWKB, ST_AsEWKT, ST_GeomFromText

7.7 Operators

7.7.1 &&

&& — Returns `TRUE` if A's bounding box overlaps B's.

Synopsis

boolean **&&**(geometry A , geometry B);
boolean **&&**(geography A , geography B);

Description

The && operator returns TRUE if the bounding box of geometry A overlaps the bounding box of geometry B.

Note
This operand will make use of any indexes that may be available on the geometries.

Availability: 1.5.0 support for geography was introduced.

✓ This method supports Circular Strings and Curves

Examples

```
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)
```

See Also

|&>, &>, &<|, &<, ~, @

7.7.2 &<

&< — Returns TRUE if A's bounding box overlaps or is to the left of B's.

Synopsis

boolean **&<**(geometry A , geometry B);

Description

The &< operator returns TRUE if the bounding box of geometry A overlaps or is to the left of the bounding box of geometry B, or more accurately, overlaps or is NOT to the right of the bounding box of geometry B.

Note
This operand will make use of any indexes that may be available on the geometries.

Examples

```
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)
```

See Also

&&, |&>, &>, &<|

7.7.3 &<|

&<| — Returns TRUE if A’s bounding box overlaps or is below B’s.

Synopsis

boolean **&<|**(geometry A , geometry B);

Description

The &<| operator returns TRUE if the bounding box of geometry A overlaps or is below of the bounding box of geometry B, or more accurately, overlaps or is NOT above the bounding box of geometry B.

✓ This method supports Circular Strings and Curves

Note
This operand will make use of any indexes that may be available on the geometries.

Examples

```
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)
```

See Also

&&, |&>, &>, &<

7.7.4 &>

&> — Returns TRUE if A’ bounding box overlaps or is to the right of B’s.

Synopsis

boolean **&>**(geometry A , geometry B);

Description

The &> operator returns TRUE if the bounding box of geometry A overlaps or is to the right of the bounding box of geometry B, or more accurately, overlaps or is NOT to the left of the bounding box of geometry B.

Note
This operand will make use of any indexes that may be available on the geometries.

Examples

```
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)
```

See Also

&&, |&>, &<|, &<

7.7.5 <<

<< — Returns TRUE if A’s bounding box is strictly to the left of B’s.

Synopsis

boolean **<<**(geometry A , geometry B);

Description

The << operator returns TRUE if the bounding box of geometry A is strictly to the left of the bounding box of geometry B.

> **Note**
> This operand will make use of any indexes that may be available on the geometries.

Examples

```
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)
```

See Also

>>, |>>, <<|

7.7.6 <<|

<<| — Returns TRUE if A’s bounding box is strictly below B’s.

Synopsis

boolean **<<|**(geometry A , geometry B);

Description

The `<<|` operator returns `TRUE` if the bounding box of geometry A is strictly below the bounding box of geometry B.

> **Note**
> This operand will make use of any indexes that may be available on the geometries.

Examples

```
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)
```

See Also

<<, >>, |>>

7.7.7 =

= — Returns `TRUE` if A’s bounding box is the same as B’s (uses float4 boxes).

Synopsis

boolean =(geometry A , geometry B);
boolean =(geography A , geography B);

Description

The = operator returns `TRUE` if the bounding box of geometry/geography A is the same as the bounding box 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).

> **Warning**
> This is cause for a lot of confusion. When you compare geometryA = geometryB it will return true even when the geometries are clearly different IF their bounding boxes are the same. To check for true equality use ST_OrderingEquals or ST_Equals. Even for points, doing a bounding box check is not sufficient to determine true equality of points since bounding box prior to PostGIS 2.0 are stored as float4.

Caution
This operand will NOT make use of any indexes that may be available on the geometries.

✓ This method supports Circular Strings and Curves

Examples

```
SELECT 'LINESTRING(0 0, 0 1, 1 0)'::geometry = 'LINESTRING(1 1, 0 0)'::geometry;
 ?column?
----------
 t
(1 row)

SELECT ST_AsText(column1)
FROM ( VALUES
  ('LINESTRING(0 0, 1 1)'::geometry),
  ('LINESTRING(1 1, 0 0)'::geometry)) AS foo;
    st_astext
---------------------
 LINESTRING(0 0,1 1)
 LINESTRING(1 1,0 0)
(2 rows)

-- 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)
(1 row)
-- NOTE: Although the points are different, the float4 boxes are the same
-- In versions 2.0+ and after, this will return false since 2.0+ switched
-- to store double-precision (float8) bounding boxes instead of float4 (used in 1.5 and ↩
    prior) --
 SELECT ST_GeomFromText('POINT(1707296.37 4820536.77)') =
  ST_GeomFromText('POINT(1707296.27 4820536.87)') As pt_intersect;
--pt_intersect --
t
```

See Also

ST_Equals, ST_OrderingEquals, ~=

7.7.8 >>

>> — Returns TRUE if A's bounding box is strictly to the right of B's.

Synopsis

boolean **>>**(geometry A , geometry B);

Description

The >> operator returns TRUE if the bounding box of geometry A is strictly to the right of the bounding box of geometry B.

> **Note**
> This operand will make use of any indexes that may be available on the geometries.

Examples

```
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)
```

See Also

<<, |>>, <<|

7.7.9 @

@ — Returns TRUE if A's bounding box is contained by B's.

Synopsis

boolean @(geometry A , geometry B);

Description

The @ operator returns TRUE if the bounding box of geometry A is completely contained by the bounding box of geometry B.

> **Note**
> This operand will make use of any indexes that may be available on the geometries.

Examples

```
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)
```

See Also

~, &&

7.7.10 |&>

|&> — Returns `TRUE` if A's bounding box overlaps or is above B's.

Synopsis

boolean **|&>**(geometry A , geometry B);

Description

The `|&>` operator returns `TRUE` if the bounding box of geometry A overlaps or is above the bounding box of geometry B, or more accurately, overlaps or is NOT below the bounding box of geometry B.

Note
This operand will make use of any indexes that may be available on the geometries.

Examples

```
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)
```

See Also

&&, &>, &<|, &<

7.7.11 |>>

|>> — Returns TRUE if A's bounding box is strictly above B's.

Synopsis

boolean **|>>**(geometry A , geometry B);

Description

The |>> operator returns TRUE if the bounding box of geometry A is strictly to the right of the bounding box of geometry B.

Note
This operand will make use of any indexes that may be available on the geometries.

Examples

```
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)
```

See Also

<<, >>, <<|

7.7.12 ~

~ — Returns TRUE if A's bounding box contains B's.

Synopsis

boolean ~(geometry A , geometry B);

Description

The ~ operator returns `TRUE` if the bounding box of geometry A completely contains the bounding box of geometry B.

Note
This operand will make use of any indexes that may be available on the geometries.

Examples

```
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)
```

See Also

@, &&

7.7.13 ~=

~= — Returns `TRUE` if A's bounding box is the same as B's.

Synopsis

boolean ~=(geometry A , geometry B);

Description

The ~= operator returns `TRUE` if the bounding box of geometry/geography A is the same as the bounding box of geometry/geography B.

Note
This operand will make use of any indexes that may be available on the geometries.

Availability: 1.5.0 changed behavior

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 and to check for bounding box equality =; operator is a safer option.

Examples

```
select ’LINESTRING(0 0, 1 1)’::geometry ~= ’LINESTRING(0 1, 1 0)’::geometry as equality;
 equality    |
-----------------+
          t     |
```

The above can be used to test if you have the new or old behavior of ~= operator.

See Also

ST_Equals, ST_OrderingEquals, =

7.8 Spatial Relationships and Measurements

7.8.1 ST_Area

ST_Area — Returns the area of the surface if it is a polygon or multi-polygon. For "geometry" type area is in SRID units. For "geography" area is in square meters.

Synopsis

float **ST_Area**(geometry g1);
float **ST_Area**(geography g1);
float **ST_Area**(geography g1, boolean use_spheroid);

Description

Returns the area of the geometry if it is a polygon or multi-polygon. Return the area measurement of an ST_Surface or ST_MultiSurface value. For geometry Area is in the units of the srid. For geography area is in square meters and defaults to measuring about the spheroid of the geography (currently only WGS84). To measure around the faster but less accurate sphere -- ST_Area(geog,false).

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

✓ This method implements the SQL/MM specification. SQL-MM 3: 8.1.2, 9.5.3

Examples

Return area in square feet for a plot of Massachusetts land and multiply by conversion to get square meters. Note this is in square feet because 2249 is Mass State Plane Feet

```
SELECT ST_Area(the_geom) As sqft, ST_Area(the_geom)*POWER(0.3048,2) As sqm
    FROM (SELECT
    ST_GeomFromText('POLYGON((743238 2967416,743238 2967450,
      743265 2967450,743265.625 2967416,743238 2967416))',2249) ) As foo(the_geom);
  sqft    |      sqm
---------+-------------
 928.625 | 86.27208552
```

Return area square feet and transform to Massachusetts state plane meters (26986) to get square meters. Note this is in square feet because 2249 is Mass State Plane Feet and transformed area is in square meters since 26986 is state plane mass meters

```

SELECT ST_Area(the_geom) As sqft, ST_Area(ST_Transform(the_geom,26986)) As sqm
    FROM (SELECT
    ST_GeomFromText('POLYGON((743238 2967416,743238 2967450,
      743265 2967450,743265.625 2967416,743238 2967416))',2249) ) As foo(the_geom);
  sqft    |        sqm
---------+------------------
 928.625 | 86.2724304199219
```

Return area square feet and square meters using Geography data type. Note that we transform to our geometry to geography (before you can do that make sure your geometry is in WGS 84 long lat 4326). Geography always measures in meters. This is just for demonstration to compare. Normally your table will be stored in geography data type already.

```

SELECT ST_Area(the_geog)/POWER(0.3048,2) As sqft_spheroid,  ST_Area(the_geog,false)/POWER ↩
    (0.3048,2) As sqft_sphere, ST_Area(the_geog) As sqm_spheroid
    FROM (SELECT
    geography(
    ST_Transform(
      ST_GeomFromText('POLYGON((743238 2967416,743238 2967450,743265 2967450,743265.625  ↩
          2967416,743238 2967416))',
        2249
        ) ,4326
      )
    )
  ) As foo(the_geog);
 sqft_spheroid   |   sqft_sphere    |   sqm_spheroid
-----------------+------------------+------------------
928.684405217197 | 927.186481558724 | 86.2776044452694

 --if your data is in geography already
 SELECT ST_Area(the_geog)/POWER(0.3048,2) As  sqft, ST_Area(the_geog) As sqm
  FROM somegeogtable;
```

See Also

ST_GeomFromText, ST_GeographyFromText, ST_SetSRID,ST_Transform

7.8.2 ST_Azimuth

ST_Azimuth — Returns the angle in radians from the horizontal of the vector defined by pointA and pointB

Synopsis

float **ST_Azimuth**(geometry pointA, geometry pointB);

Description

Returns the azimuth of the segment defined by the given Point geometries, or NULL if the two points are coincident. Return value is in radians.

The Azimuth is mathematical concept defined as the angle, in this case measured in radian, between a reference plane and a point

Availability: 1.1.0

Azimuth is especially useful in conjunction with ST_Translate for shifting an object along its perpendicular axis. See upgis_lineshift Plpgsqlfunctions PostGIS wiki section for example of this.

Examples

--Azimuth in degrees

```
SELECT ST_Azimuth(ST_MakePoint(1,2), ST_MakePoint(3,4))/(2*pi())*360 as degAz,
  ST_Azimuth(ST_MakePoint(3,4), ST_MakePoint(1,2))/(2*pi())*360 As degAzrev

degaz degazrev
------  ----------
45    225
```

See Also

ST_MakePoint, ST_Translate

7.8.3 ST_Centroid

ST_Centroid — Returns the geometric center of a geometry.

Synopsis

geometry **ST_Centroid**(geometry g1);

Description

Computes the geometric center of a geometry, or equivalently, the center of mass of the geometry as a POINT. For [MULTI]POINTs, this is computed as the arithmetic mean of the input coordinates. For [MULTI]LINESTRINGs, this is computed as the weighted length of each line segment. For [MULTI]POLYGONs, "weight" is thought in terms of area. If an empty geometry is supplied, an empty GEOMETRYCOLLECTION is returned. If NULL is supplied, NULL is returned.

The centroid is equal to the centroid of the set of component Geometries of highest dimension (since the lower-dimension geometries contribute zero "weight" to the centroid).

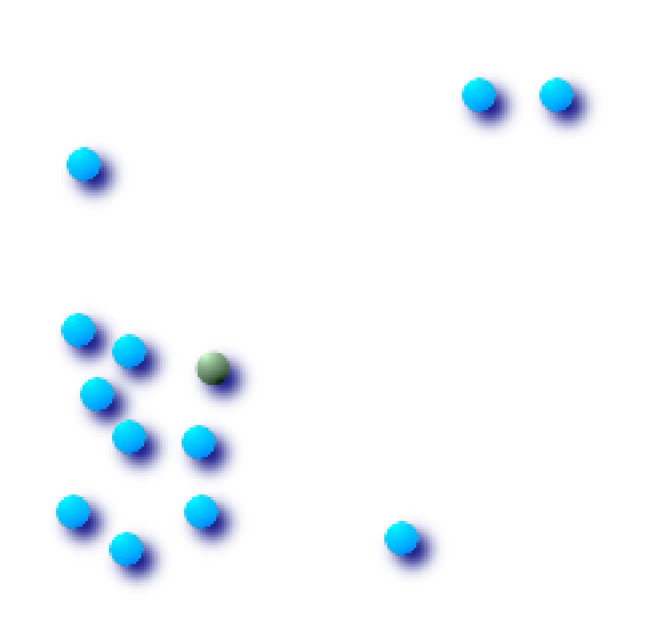
Note
Computation will be more accurate if performed by the GEOS module (enabled at compile time).

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

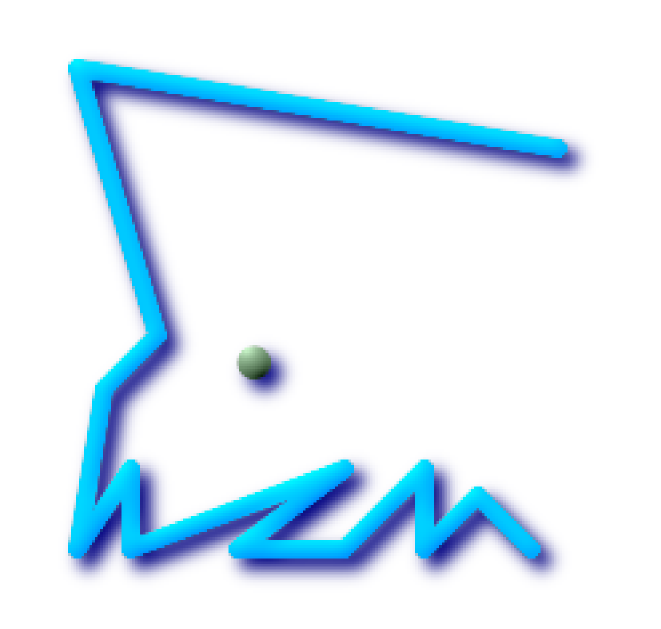
✓ This method implements the SQL/MM specification. SQL-MM 3: 8.1.4, 9.5.5

Examples

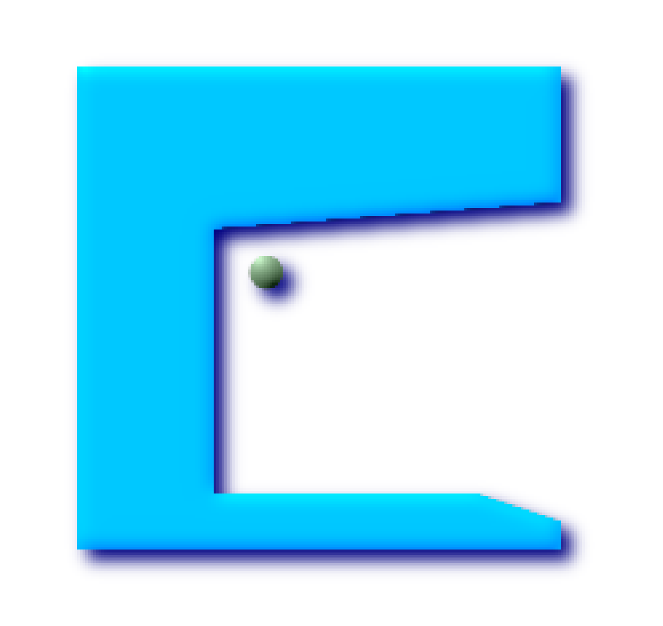
In each of the following illustrations, the blue dot represents the centroid of the source geometry.



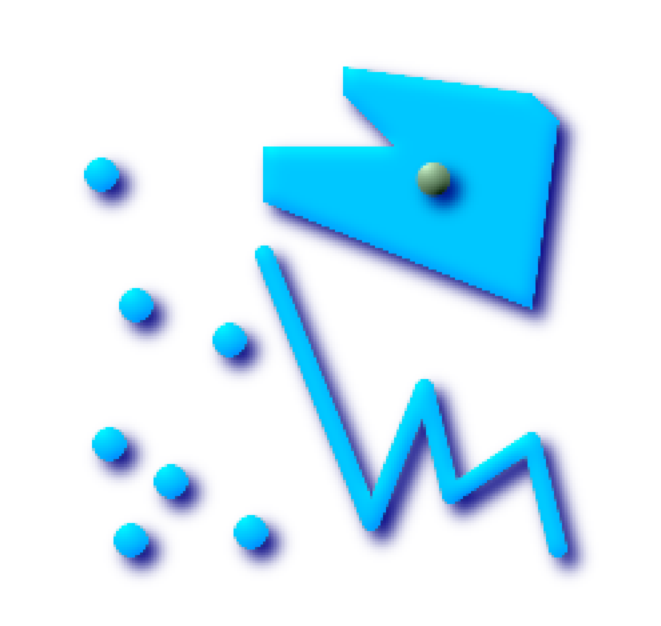
Centroid of a `MULTIPOINT`



Centroid of a `LINESTRING`



Centroid of a `POLYGON`



Centroid of a `GEOMETRYCOLLECTION`

```
SELECT ST_AsText(ST_Centroid('MULTIPOINT ( -1 0, -1 2, -1 3, -1 4, -1 7, 0 1, 0 3, 1 1, 2  ↩
    0, 6 0, 7 8, 9 8, 10 6 )'));
        st_astext
------------------------------------------
 POINT(2.30769230769231 3.30769230769231)
(1 row)
```

See Also

ST_PointOnSurface

7.8.4 ST_ClosestPoint

ST_ClosestPoint — Returns the 2-dimensional point on g1 that is closest to g2. This is the first point of the shortest line.

Synopsis

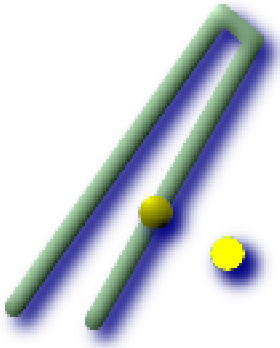
geometry **ST_ClosestPoint**(geometry g1, geometry g2);

Description

Returns the 2-dimensional point on g1 that is closest to g2. This is the first point of the shortest line.

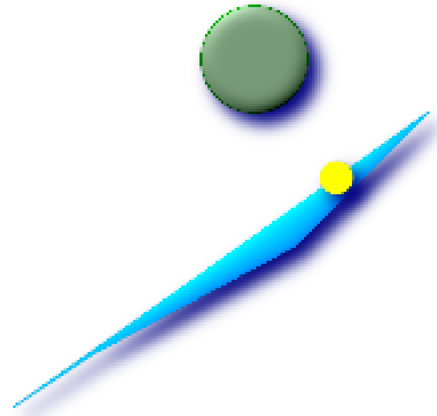
Availability: 1.5.0

Examples



Closest between point and linestring is the point itself, but closest point between a linestring and point is the point on line string that is closest.

```
SELECT ST_AsText(ST_ClosestPoint(pt,line)) AS cp_pt_line,
        ST_AsText(ST_ClosestPoint(line,pt)) As cp_line_pt
FROM (SELECT 'POINT(100 100)'::geometry As pt,
            'LINESTRING (20 80, 98 190, 110 180, 50 75 )'::geometry As line
        ) As foo;


   cp_pt_line   |                cp_line_pt
----------------+------------------------------------------
 POINT(100 100) | POINT(73.0769230769231 115.384615384615)
```



closest point on polygon A to polygon B

```
SELECT ST_AsText(
                ST_ClosestPoint(
                        ST_GeomFromText('POLYGON((175 150, 20 40, 50 60, 125 100, 175 150
                        ST_Buffer(ST_GeomFromText('POINT(110 170)'), 20)
                        )
                ) As ptwkt;

                  ptwkt
------------------------------------------
 POINT(140.752120669087 125.695053378061)
```

See Also

ST_Distance, ST_LongestLine, ST_ShortestLine, ST_MaxDistance

7.8.5 ST_Contains

ST_Contains — Returns true if and only if no points of B lie in the exterior of A, and at least one point of the interior of B lies in the interior of A.

Synopsis

boolean **ST_Contains**(geometry geomA, geometry geomB);

Description

Geometry A contains Geometry B if and only if no points of B lie in the exterior of A, and at least one point of the interior of B lies in the interior of A. An important subtlety of this definition is that A does not contain its boundary, but A does contain itself. Contrast that to ST_ContainsProperly where geometry A does not Contain Properly itself.

Returns TRUE if geometry B is completely inside geometry A. For this function to make sense, the source geometries must both be of the same coordinate projection, having the same SRID. ST_Contains is the inverse of ST_Within. So ST_Contains(A,B) implies ST_Within(B,A) except in the case of invalid geometries where the result is always false regardless or not defined.

Performed by the GEOS module

Important
Do not call with a `GEOMETRYCOLLECTION` as an argument

Important
Do not use this function with invalid geometries. You will get unexpected results.

This function call will automatically include a bounding box comparison that will make use of any indexes that are available on the geometries. To avoid index use, use the function _ST_Contains.

NOTE: this is the "allowable" version that returns a boolean, not an integer.

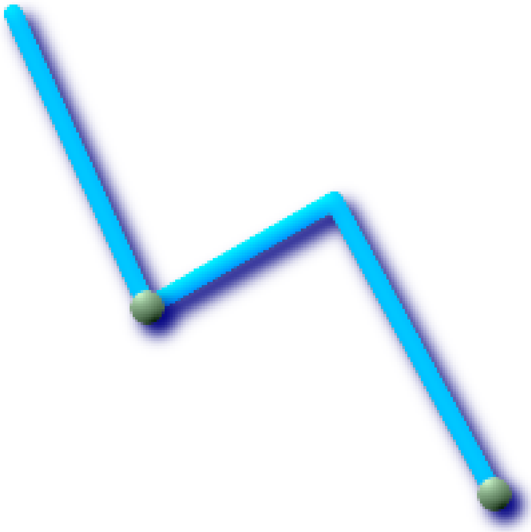
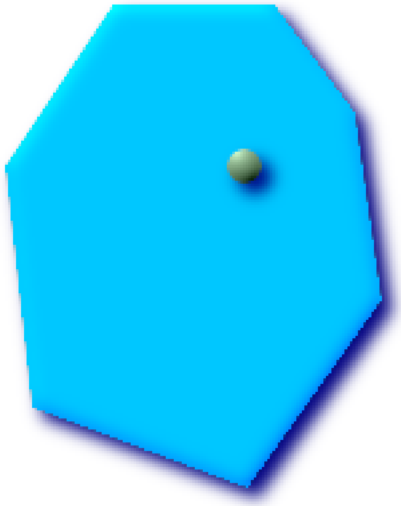
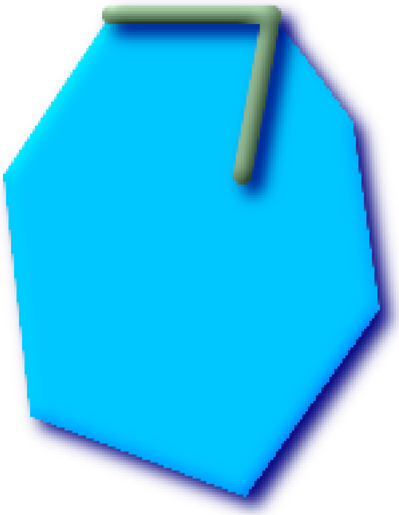
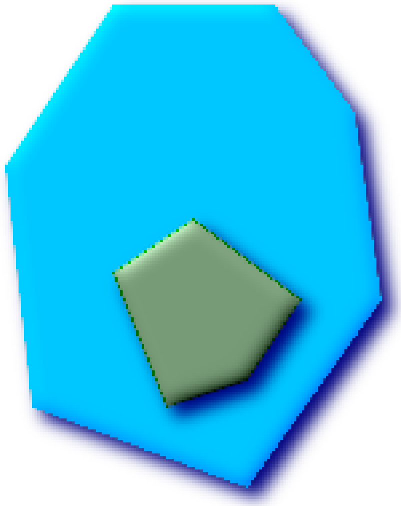
✓ This method implements the OpenGIS Simple Features Implementation Specification for SQL 1.1. s2.1.1.2 // s2.1.13.3 - same as within(geometry B, geometry A)

✓ This method implements the SQL/MM specification. SQL-MM 3: 5.1.31

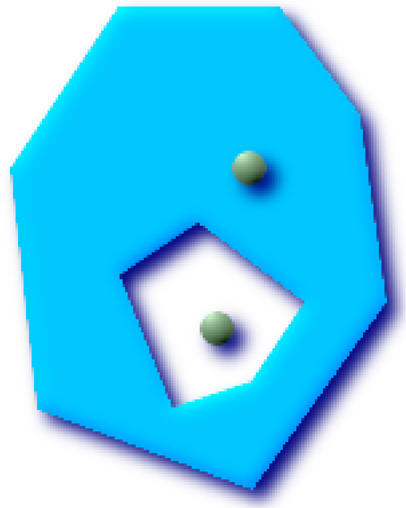
There are certain subtleties to ST_Contains and ST_Within that are not intuitively obvious. For details check out Subtleties of OGC Covers, Contains, Within

Examples

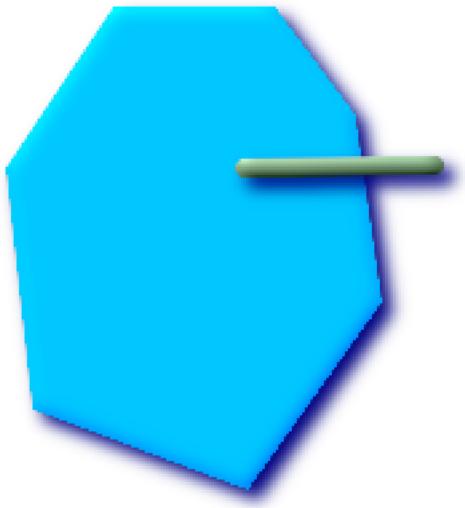
The `ST_Contains` predicate returns `TRUE` in all the following illustrations.

 *LINESTRING / MULTIPOINT*	 *POLYGON / POINT*
 *POLYGON / LINESTRING*	 *POLYGON / POLYGON*

The `ST_Contains` predicate returns `FALSE` in all the following illustrations.



POLYGON / MULTIPOINT



POLYGON / LINESTRING

```
-- A circle within a circle
SELECT ST_Contains(smallc, bigc) As smallcontainsbig,
     ST_Contains(bigc,smallc) As bigcontainssmall,
     ST_Contains(bigc, ST_Union(smallc, bigc)) as bigcontainsunion,
     ST_Equals(bigc, ST_Union(smallc, bigc)) as bigisunion,
     ST_Covers(bigc, ST_ExteriorRing(bigc)) As bigcoversexterior,
     ST_Contains(bigc, ST_ExteriorRing(bigc)) As bigcontainsexterior
FROM (SELECT ST_Buffer(ST_GeomFromText('POINT(1 2)'), 10) As smallc,
       ST_Buffer(ST_GeomFromText('POINT(1 2)'), 20) As bigc) As foo;

-- Result
  smallcontainsbig | bigcontainssmall | bigcontainsunion | bigisunion | bigcoversexterior | ↩
       bigcontainsexterior
------------------+------------------+------------------+------------+-------------------+----------

 f                | t                | t                | t          | t          | f

-- Example demonstrating difference between contains and contains properly
SELECT ST_GeometryType(geomA) As geomtype, ST_Contains(geomA,geomA) AS acontainsa,  ↩
    ST_ContainsProperly(geomA, geomA) AS acontainspropa,
   ST_Contains(geomA, ST_Boundary(geomA)) As acontainsba, ST_ContainsProperly(geomA,  ↩
       ST_Boundary(geomA)) As acontainspropba
FROM (VALUES ( ST_Buffer(ST_Point(1,1), 5,1) ),
        ( ST_MakeLine(ST_Point(1,1), ST_Point(-1,-1) ) ),
        ( ST_Point(1,1) )
    ) As foo(geomA);

  geomtype    | acontainsa | acontainspropa | acontainsba | acontainspropba
--------------+------------+----------------+-------------+-----------------
ST_Polygon    | t          | f              | f           | f
ST_LineString | t          | f              | f           | f
ST_Point      | t          | t              | f           | f
```

See Also

ST_Boundary, ST_ContainsProperly, ST_Covers,ST_CoveredBy, ST_Equals,ST_Within

7.8.6 ST_ContainsProperly

ST_ContainsProperly — Returns true if B intersects the interior of A but not the boundary (or exterior). A does not contain properly itself, but does contain itself.

Synopsis

boolean **ST_ContainsProperly**(geometry geomA, geometry geomB);

Description

Returns true if B intersects the interior of A but not the boundary (or exterior).

A does not contain properly itself, but does contain itself.

Every point of the other geometry is a point of this geometry's interior. The DE-9IM Intersection Matrix for the two geometries matches [T**FF*FF*] used in ST_Relate

Note
From JTS docs slightly reworded: The advantage to using this predicate over ST_Contains and ST_Intersects is that it can be computed efficiently, with no need to compute topology at individual points.
An example use case 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 wholly inside the area. In these cases the intersection is known a priori to be exactly the original test geometry.

Availability: 1.4.0 - requires GEOS >= 3.1.0.

Important
Do not call with a `GEOMETRYCOLLECTION` as an argument

Important
Do not use this function with invalid geometries. You will get unexpected results.

This function call will automatically include a bounding box comparison that will make use of any indexes that are available on the geometries. To avoid index use, use the function _ST_ContainsProperly.

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

7.8.7 ST_Covers

ST_Covers — Returns 1 (TRUE) if no point in Geometry B is outside Geometry A. For geography: if geography point B is not outside Polygon Geography A

Synopsis

boolean **ST_Covers**(geometry geomA, geometry geomB);
boolean **ST_Covers**(geography geogpolyA, geography geogpointB);

Description

Returns 1 (TRUE) if no point in Geometry/Geography B is outside Geometry/Geography A

Performed by the GEOS module

Important
Do not call with a GEOMETRYCOLLECTION as an argument

Important
For geography only Polygon covers point is supported.

Important
Do not use this function with invalid geometries. You will get unexpected results.

This function call will automatically include a bounding box comparison that will make use of any indexes that are available on the geometries. To avoid index use, use the function _ST_Covers.

Availability: 1.2.2 - requires GEOS >= 3.0

Availability: 1.5 - support for geography was introduced.

NOTE: this is the "allowable" version that returns a boolean, not an integer.

Not an OGC standard, but Oracle has it too.

There are certain subtleties to ST_Contains and ST_Within that are not intuitively obvious. For details check out Subtleties of OGC Covers, Contains, Within

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

7.8.8 ST_CoveredBy

ST_CoveredBy — Returns 1 (TRUE) if no point in Geometry/Geography A is outside Geometry/Geography B

Synopsis

boolean **ST_CoveredBy**(geometry geomA, geometry geomB);
boolean **ST_CoveredBy**(geography geogA, geography geogB);

Description

Returns 1 (TRUE) if no point in Geometry/Geography A is outside Geometry/Geography B

Performed by the GEOS module

Important
Do not call with a `GEOMETRYCOLLECTION` as an argument

Important
Do not use this function with invalid geometries. You will get unexpected results.

Availability: 1.2.2 - requires GEOS >= 3.0

This function call will automatically include a bounding box comparison that will make use of any indexes that are available on the geometries. To avoid index use, use the function _ST_CoveredBy.

NOTE: this is the "allowable" version that returns a boolean, not an integer.

Not an OGC standard, but Oracle has it too.

There are certain subtleties to ST_Contains and ST_Within that are not intuitively obvious. For details check out Subtleties of OGC Covers, Contains, Within

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

7.8.9 ST_Crosses

ST_Crosses — Returns `TRUE` if the supplied geometries have some, but not all, interior points in common.

Synopsis

boolean **ST_Crosses**(geometry g1, geometry g2);

Description

`ST_Crosses` takes two geometry objects and returns `TRUE` if their intersection "spatially cross", that is, the geometries have some, but not all interior points in common. The intersection of the interiors of the geometries must not be the empty set and must have a dimensionality less than the the maximum dimension of the two input geometries. Additionally, the intersection of the two geometries must not equal either of the source geometries. Otherwise, it returns `FALSE`.

In mathematical terms, this is expressed as:

$$a.Crosses(b) \Leftrightarrow (dim(I(a) \cap I(b)) < max(dim(I(a)), dim(I(b)))) \wedge (a \cap b \neq a) \wedge (a \cap b \neq b)$$

The DE-9IM Intersection Matrix for the two geometries is:

- 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)

For any other combination of dimensions this predicate returns false.

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.

Important
Do not call with a `GEOMETRYCOLLECTION` as an argument

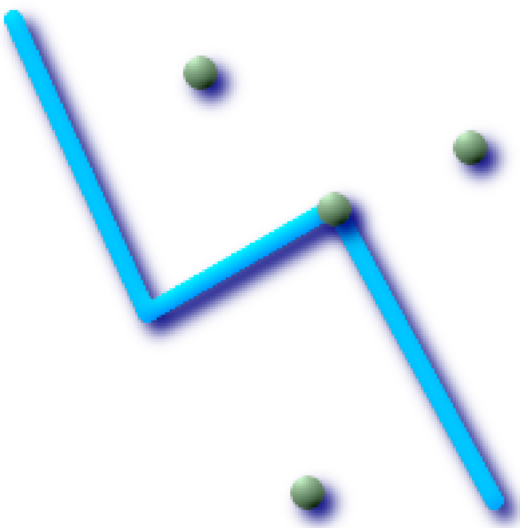
Note
This function call will automatically include a bounding box comparison that will make use of any indexes that are available on the geometries.

✓ This method implements the OpenGIS Simple Features Implementation Specification for SQL 1.1. s2.1.13.3

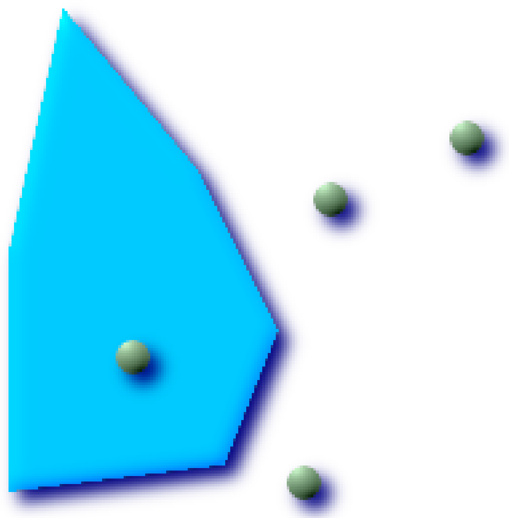
✓ This method implements the SQL/MM specification. SQL-MM 3: 5.1.29

Examples

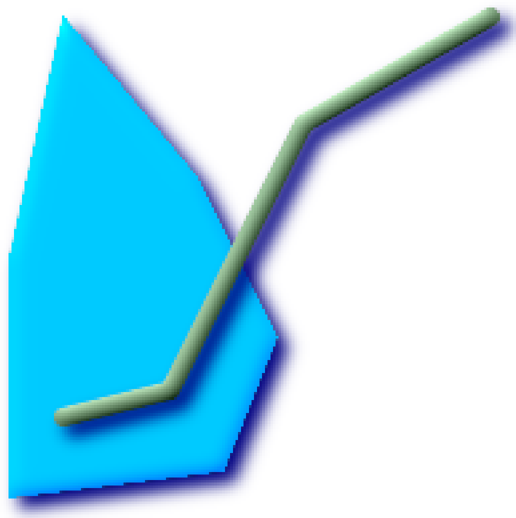
The following illustrations all return `TRUE`.



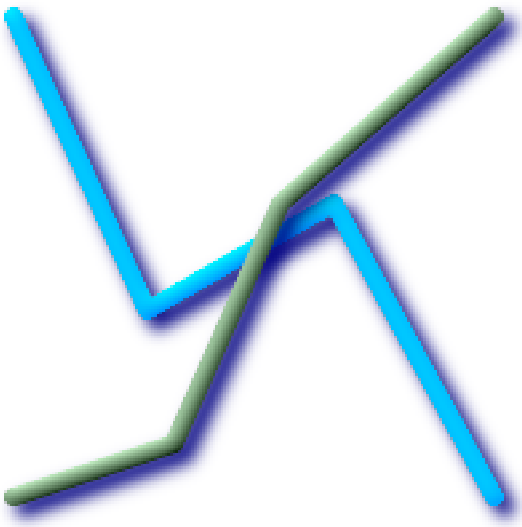
MULTIPOINT / LINESTRING



MULTIPOINT / POLYGON



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,
  the_geom geometry,
  CONSTRAINT roads_pkey PRIMARY KEY ( ←↩
    road_id)
);
```

```
CREATE TABLE highways (
  id serial NOT NULL,
  the_gem geometry,
  CONSTRAINT roads_pkey PRIMARY KEY ( ←↩
    road_id)
);
```

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.the_geom, highways.the_geom);
```

7.8.10 ST_LineCrossingDirection

ST_LineCrossingDirection — Given 2 linestrings, returns a number between -3 and 3 denoting what kind of crossing behavior. 0 is no crossing.

Synopsis

integer **ST_LineCrossingDirection**(geometry linestringA, geometry linestringB);

Description

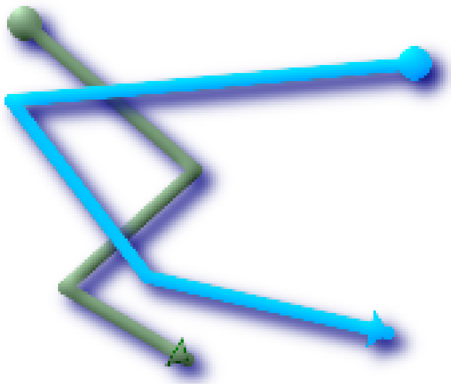
Given 2 linestrings, returns a number between -3 and 3 denoting what kind of crossing behavior. 0 is no crossing. This is only supported for `LINESTRING`

Definition of integer constants is as follows:

- 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

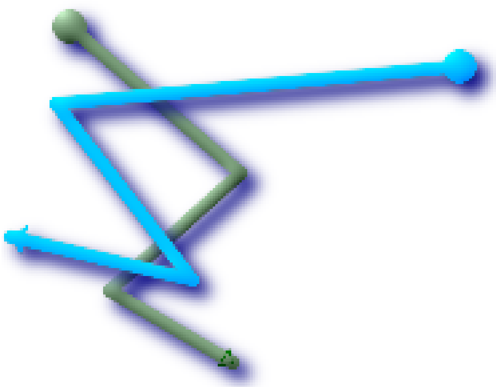
Availability: 1.4

Examples



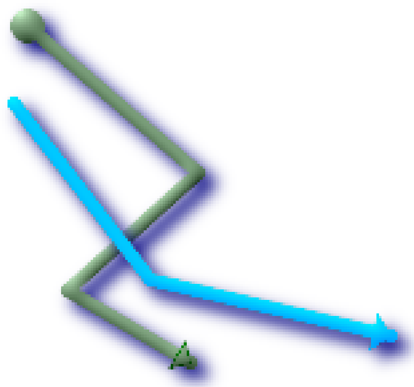
Line 1 (green), Line 2 ball is start point, triangle are end points. Query below.

```
SELECT ST_LineCrossingDirection(foo.line1 , foo.line2) As l1_cross_l2 ,
          ST_LineCrossingDirection(foo.line2, foo.line1) As l2_cross_l1
FROM (
SELECT
 ST_GeomFromText(’LINESTRING(25 169,89 114,40 70,86 43)’) As line1,
 ST_GeomFromText(’LINESTRING(171 154,20 140,71 74,161 53)’) As line2
        ) As foo;

 l1_cross_l2 | l2_cross_l1
-------------+-------------
           3 |          -3
```



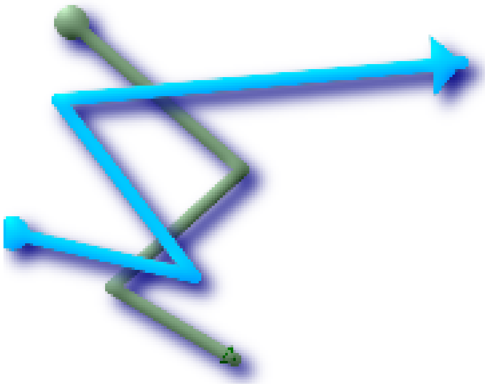
Line 1 (green), Line 2 (blue) ball is start point, triangle are end points. Query below.

```
SELECT ST_LineCrossingDirection(foo.line1 , foo.line2) As l1_cross_l2 ,
          ST_LineCrossingDirection(foo.line2, foo.line1) As l2_cross_l1
FROM (
 SELECT
  ST_GeomFromText(’LINESTRING(25 169,89 114,40 70,86 43)’) As line1,
  ST_GeomFromText(’LINESTRING (171 154, 20 140, 71 74, 2.99 90.16)’) As line2
) As foo;

 l1_cross_l2 | l2_cross_l1
-------------+-------------
           2 |          -2
```



Line 1 (green), Line 2 (blue) ball is start point, triangle are end points. Query below.

```
SELECT
        ST_LineCrossingDirection(foo. ↩
    line1, foo.line2) As l1_cross_l2 ,
        ST_LineCrossingDirection(foo. ↩
    line2, foo.line1) As l2_cross_l1
FROM (
 SELECT
  ST_GeomFromText('LINESTRING(25 169,89  ↩
    114,40 70,86 43)') As line1,
  ST_GeomFromText('LINESTRING (20 140, 71 ↩
     74, 161 53)') As line2
  ) As foo;

 l1_cross_l2 | l2_cross_l1
-------------+-------------
          -1 |           1
```



Line 1 (green), Line 2 (blue) ball is start point, triangle are end points. Query below.

```
SELECT ST_LineCrossingDirection(foo.line1 ↩
    , foo.line2) As l1_cross_l2 ,
        ST_LineCrossingDirection(foo. ↩
    line2, foo.line1) As l2_cross_l1
FROM (SELECT
        ST_GeomFromText('LINESTRING(25  ↩
    169,89 114,40 70,86 43)') As line1,
        ST_GeomFromText('LINESTRING(2.99  ↩
    90.16,71 74,20 140,171 154)') As line2
        ) As foo;

 l1_cross_l2 | l2_cross_l1
-------------+-------------
          -2 |           2
```

```
SELECT s1.gid, s2.gid, ST_LineCrossingDirection(s1.the_geom, s2.the_geom)
  FROM streets s1 CROSS JOIN streets s2 ON (s1.gid != s2.gid AND s1.the_geom && s2.the_geom ↩
       )
WHERE ST_CrossingDirection(s1.the_geom, s2.the_geom) > 0;
```

See Also

ST_Crosses

7.8.11 ST_Disjoint

ST_Disjoint — Returns TRUE if the Geometries do not "spatially intersect" - if they do not share any space together.

Synopsis

boolean **ST_Disjoint**(geometry A , geometry B);

Description

Overlaps, Touches, Within all imply geometries are not spatially disjoint. If any of the aforementioned returns true, then the geometries are not spatially disjoint. Disjoint implies false for spatial intersection.

Important
Do not call with a GEOMETRYCOLLECTION as an argument

Performed by the GEOS module

Note
This function call does not use indexes

Note
NOTE: this is the "allowable" version that returns a boolean, not an integer.

✓ This method implements the OpenGIS 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_IntersectsST_Intersects

7.8.12 ST_Distance

ST_Distance — For geometry type Returns the 2-dimensional cartesian minimum distance (based on spatial ref) between two geometries in projected units. For geography type defaults to return spheroidal minimum distance between two geographies in meters.

Synopsis

float **ST_Distance**(geometry g1, geometry g2);
float **ST_Distance**(geography gg1, geography gg2);
float **ST_Distance**(geography gg1, geography gg2, boolean use_spheroid);

Description

For geometry type returns the 2-dimensional minimum cartesian distance between two geometries in projected units (spatial ref units). For geography type defaults to return the minimum distance around WGS 84 spheroid between two geographies in meters. Pass in false to return answer in sphere instead of spheroid.

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

✓ This method implements the SQL/MM specification. SQL-MM 3: 5.1.23

Availability: 1.5.0 geography support was introduced in 1.5. Speed improvements for planar to better handle large or many vertex geometries

Examples

```
--Geometry example - units in planar degrees 4326 is WGS 84 long lat unit=degrees
SELECT ST_Distance(
    ST_GeomFromText('POINT(-72.1235 42.3521)',4326),
    ST_GeomFromText('LINESTRING(-72.1260 42.45, -72.123 42.1546)', 4326)
  );
st_distance
-----------------
0.00150567726382282

-- Geometry example - units in meters (SRID: 26986 Massachusetts state plane meters) (most  ↩
    accurate for Massachusetts)
SELECT ST_Distance(
      ST_Transform(ST_GeomFromText('POINT(-72.1235 42.3521)',4326),26986),
      ST_Transform(ST_GeomFromText('LINESTRING(-72.1260 42.45, -72.123 42.1546)', 4326) ↩
          ,26986)
    );
st_distance
-----------------
123.797937878454

-- Geometry example - units in meters (SRID: 2163 US National Atlas Equal area) (least  ↩
    accurate)
SELECT ST_Distance(
      ST_Transform(ST_GeomFromText('POINT(-72.1235 42.3521)',4326),2163),
      ST_Transform(ST_GeomFromText('LINESTRING(-72.1260 42.45, -72.123 42.1546)', 4326) ↩
          ,2163)
    );

st_distance
------------------
```

```
126.664256056812

-- Geography example -- same but note units in meters - use sphere for slightly faster less ↩
    accurate
SELECT ST_Distance(gg1, gg2) As spheroid_dist, ST_Distance(gg1, gg2, false) As sphere_dist
FROM (SELECT
  ST_GeographyFromText('SRID=4326;POINT(-72.1235 42.3521)') As gg1,
  ST_GeographyFromText('SRID=4326;LINESTRING(-72.1260 42.45, -72.123 42.1546)') As gg2
  ) As foo  ;

  spheroid_dist    |   sphere_dist
------------------+------------------
 123.802076746848 | 123.475736916397
```

See Also

ST_DWithin, ST_Distance_Sphere, ST_Distance_Spheroid, ST_MaxDistance, ST_Transform

7.8.13 ST_HausdorffDistance

ST_HausdorffDistance — Returns the Hausdorff distance between two geometries. Basically a measure of how similar or dissimilar 2 geometries are. Units are in the units of the spatial reference system of the geometries.

Synopsis

float **ST_HausdorffDistance**(geometry g1, geometry g2);
float **ST_HausdorffDistance**(geometry g1, geometry g2, float densifyFrac);

Description

Implements algorithm for computing a distance metric which can be thought of as the "Discrete Hausdorff Distance". This is the Hausdorff distance restricted to discrete points for one of the geometries. Wikipedia article on Hausdorff distance Martin Davis note on how Hausdorff Distance calculation was used to prove correctness of the CascadePolygonUnion approach.

When densifyFrac is specified, this function performs a segment densification before computing the discrete hausdorff 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.

Note
The current implementation supports only vertices as the discrete locations. This could be extended to allow an arbitrary density of points to be used.

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 part of this subset is Linestrings that are roughly parallel to each other, and roughly equal in length. This is a useful metric for line matching.

Availability: 1.5.0 - requires GEOS >= 3.2.0

Examples

```
postgis=# SELECT st_HausdorffDistance(
        'LINESTRING (0 0, 2 0)'::geometry,
        'MULTIPOINT (0 1, 1 0, 2 1)'::geometry);
 st_hausdorffdistance
----------------------
           1
(1 row)
```

```
postgis=# SELECT st_hausdorffdistance('LINESTRING (130 0, 0 0, 0 150)'::geometry, ' ↩
    LINESTRING (10 10, 10 150, 130 10)'::geometry, 0.5);
 st_hausdorffdistance
----------------------
          70
(1 row)
```

7.8.14 ST_MaxDistance

ST_MaxDistance — Returns the 2-dimensional largest distance between two geometries in projected units.

Synopsis

float **ST_MaxDistance**(geometry g1, geometry g2);

Description

Some useful description here.

Note
Returns the 2-dimensional maximum distance between two linestrings in projected units. If g1 and g2 is the same geometry the function will return the distance between the two vertices most far from each other in that geometry.

Availability: 1.5.0

Examples

```
postgis=# SELECT ST_MaxDistance('POINT(0 0)'::geometry, 'LINESTRING ( 2 0, 0 2 )'::geometry ↩
    );
   st_maxdistance
-----------------
 2
(1 row)

postgis=# SELECT ST_MaxDistance('POINT(0 0)'::geometry, 'LINESTRING ( 2 2, 2 2 )'::geometry ↩
    );
  st_maxdistance
------------------
 2.82842712474619
(1 row)
```

See Also

ST_Distance, ST_LongestLine

7.8.15 ST_Distance_Sphere

ST_Distance_Sphere — Returns minimum distance in meters between two lon/lat geometries. Uses a spherical earth and radius of 6370986 meters. Faster than ST_Distance_Spheroid, but less accurate. PostGIS versions prior to 1.5 only implemented for points.

Synopsis

float **ST_Distance_Sphere**(geometry geomlonlatA, geometry geomlonlatB);

Description

Returns minimum distance in meters between two lon/lat points. Uses a spherical earth and radius of 6370986 meters. Faster than ST_Distance_Spheroid, but less accurate. PostGIS Versions prior to 1.5 only implemented for points.

Note
This function currently does not look at the SRID of a geometry and will always assume its in WGS 84 long lat. Prior versions of this function only support points.

Availability: 1.5 - support for other geometry types besides points was introduced. Prior versions only work with points.

Examples

```
SELECT round(CAST(ST_Distance_Sphere(ST_Centroid(the_geom), ST_GeomFromText('POINT(-118 38) ↩
    ',4326)) As numeric),2) As dist_meters,
round(CAST(ST_Distance(ST_Transform(ST_Centroid(the_geom),32611),
    ST_Transform(ST_GeomFromText('POINT(-118 38)', 4326),32611)) As numeric),2) As  ↩
        dist_utm11_meters,
round(CAST(ST_Distance(ST_Centroid(the_geom), ST_GeomFromText('POINT(-118 38)', 4326)) As  ↩
    numeric),5) As dist_degrees,
round(CAST(ST_Distance(ST_Transform(the_geom,32611),
    ST_Transform(ST_GeomFromText('POINT(-118 38)', 4326),32611)) As numeric),2) As  ↩
        min_dist_line_point_meters
FROM
  (SELECT ST_GeomFromText('LINESTRING(-118.584 38.374,-118.583 38.5)', 4326) As the_geom)  ↩
     as foo;
   dist_meters | dist_utm11_meters | dist_degrees | min_dist_line_point_meters
  -------------+-------------------+--------------+----------------------------
    70424.47 |          70438.00 |      0.72900 |                   65871.18
```

See Also

ST_Distance, ST_Distance_Spheroid

7.8.16 ST_Distance_Spheroid

ST_Distance_Spheroid — Returns the minimum distance between two lon/lat geometries given a particular spheroid. PostGIS versions prior to 1.5 only support points.

Synopsis

float **ST_Distance_Spheroid**(geometry geomlonlatA, geometry geomlonlatB, spheroid measurement_spheroid);

Description

Returns minimum distance in meters between two lon/lat geometries given a particular spheroid. See the explanation of spheroids given for ST_Length_Spheroid. PostGIS version prior to 1.5 only support points.

Note
This function currently does not look at the SRID of a geometry and will always assume its represented in the coordinates of the passed in spheroid. Prior versions of this function only support points.

Availability: 1.5 - support for other geometry types besides points was introduced. Prior versions only work with points.

Examples

```
SELECT round(CAST(
    ST_Distance_Spheroid(ST_Centroid(the_geom), ST_GeomFromText('POINT(-118 38)',4326), ' ↩
        SPHEROID["WGS 84",6378137,298.257223563]')
      As numeric),2) As dist_meters_spheroid,
    round(CAST(ST_Distance_Sphere(ST_Centroid(the_geom), ST_GeomFromText('POINT(-118 38) ↩
        ',4326)) As numeric),2) As dist_meters_sphere,
round(CAST(ST_Distance(ST_Transform(ST_Centroid(the_geom),32611),
    ST_Transform(ST_GeomFromText('POINT(-118 38)', 4326),32611)) As numeric),2) As  ↩
        dist_utm11_meters
FROM
  (SELECT ST_GeomFromText('LINESTRING(-118.584 38.374,-118.583 38.5)', 4326) As the_geom)  ↩
      as foo;
 dist_meters_spheroid | dist_meters_sphere | dist_utm11_meters
----------------------+--------------------+-------------------
       70454.92 |           70424.47 |          70438.00
```

See Also

ST_Distance, ST_Distance_Sphere

7.8.17 ST_DFullyWithin

ST_DFullyWithin — Returns true if all of the geometries are within the specified distance of one another

Synopsis

boolean **ST_DFullyWithin**(geometry g1, geometry g2, double precision distance);

Description

Returns true if the geometries is 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 call will automatically include a bounding box comparison that will make use of any 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

7.8.18 ST_DWithin

ST_DWithin — Returns true if the geometries are within the specified distance of one another. For geometry units are in those of spatial reference and For geography units are in meters and measurement is defaulted to use_spheroid=true (measure around spheroid), for faster check, use_spheroid=false to measure along sphere.

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 **ST_DWithin**(geography gg1, geography gg2, double precision distance_meters, boolean use_spheroid);

Description

Returns true if the geometries are within the specified distance of one another.

For Geometries: 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 coorindate projection, having the same SRID.

For geography units are in meters and measurement is defaulted to use_spheroid=true (measure around WGS 84 spheroid), for faster check, use_spheroid=false to measure along sphere.

Note
This function call will automatically include a bounding box comparison that will make use of any indexes that are available on the geometries.

Note
Prior to 1.3, ST_Expand was commonly used in conjunction with && and ST_Distance to achieve the same effect and in pre-1.3.4 this function was basically short-hand for that construct. From 1.3.4, ST_DWithin uses a more short-circuit distance function which should make it more efficient than prior versions for larger buffer regions.

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

Availability: 1.5.0 support for geography was introduced

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.the_geom, h.hospital_name
  FROM schools s
    LEFT JOIN hospitals h ON ST_DWithin(s.the_geom, h.the_geom, 3000)
  ORDER BY s.gid, ST_Distance(s.the_geom, h.the_geom);

--The schools with no close hospitals
--Find all schools with no hospital within 3000 units
--away from the school.  Units is in units of spatial ref (e.g. meters, feet, degrees)
SELECT s.gid, s.school_name
  FROM schools s
    LEFT JOIN hospitals h ON ST_DWithin(s.the_geom, h.the_geom, 3000)
  WHERE h.gid IS NULL;
```

See Also

ST_Distance, ST_Expand

7.8.19 ST_Equals

ST_Equals — Returns true if the given geometries represent the same geometry. Directionality is ignored.

Synopsis

boolean **ST_Equals**(geometry A, geometry B);

Description

Returns TRUE if the given Geometries are "spatially equal". Use this for a ’better’ answer than ’=’. Note by spatially equal we mean ST_Within(A,B) = true and ST_Within(B,A) = true and also mean ordering of points can be different but represent the same geometry structure. 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).

Important
This function will return false if either geometry is invalid even if they are binary equal.

This method implements the OpenGIS 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

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

7.8.20 ST_HasArc

ST_HasArc — Returns true if a geometry or geometry collection contains a circular string

Synopsis

boolean **ST_HasArc**(geometry geomA);

Description

Returns true if a geometry or geometry collection contains a circular string

Availability: 1.2.3?

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

This method supports Circular Strings and Curves

Examples

```
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
```

See Also

ST_CurveToLine,ST_LineToCurve

7.8.21 ST_Intersects

ST_Intersects — Returns TRUE if the Geometries/Geography "spatially intersect" - (share any portion of space) and FALSE if they don't (they are Disjoint). For geography -- tolerance is 0.00001 meters (so any points that close are considered to intersect)

Synopsis

boolean **ST_Intersects**(geometry geomA , geometry geomB);
boolean **ST_Intersects**(geography geogA , geography geogB);

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.

Important
Do not call with a `GEOMETRYCOLLECTION` as an argument for geometry version. The geography version supports GEOMETRYCOLLECTION since its a thin wrapper around distance implementation.

Performed by the GEOS module (for geometry), geography is native

Availability: 1.5 support for geography was introduced.

Note
This function call will automatically include a bounding box comparison that will make use of any indexes that are available on the geometries.

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 OpenGIS Simple Features Implementation Specification for SQL 1.1. s2.1.1.2 //s2.1.13.3 - ST_Intersects(g1, g2) --> Not (ST_Disjoint(g1, g2))

✓ This method implements the SQL/MM specification. SQL-MM 3: 5.1.27

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)
```

Geography Examples

```
SELECT ST_Intersects(
    ST_GeographyFromText('SRID=4326;LINESTRING(-43.23456 72.4567,-43.23456 72.4568)'),
    ST_GeographyFromText('SRID=4326;POINT(-43.23456 72.4567772)')
    );

 st_intersects
---------------
t
```

See Also

ST_Disjoint

7.8.22 ST_Length

ST_Length — Returns the 2d length of the geometry if it is a linestring or multilinestring. geometry are in units of spatial reference and geography are in meters (default spheroid)

Synopsis

float **ST_Length**(geometry a_2dlinestring);
float **ST_Length**(geography gg);
float **ST_Length**(geography gg, boolean use_spheroid);

Description

For geometry: Returns the cartesian 2D length of the geometry if it is a linestring, multilinestring, ST_Curve, ST_MultiCurve. 0 is returned for areal geometries. For areal geometries use ST_Perimeter. Geometry: Measurements are in the units of the spatial reference system of the geometry. Geography: Units are in meters and also acts as a Perimeter function for areal geogs.

Currently for geometry this is an alias for ST_Length2D, but this may change to support higher dimensions.

Note
Currently applying this to a MULTI/POLYGON of type geography will give you the perimeter of the POLYGON/MULTIPOLYGON. This is not the case with the geometry implementation.

Note
For geography measurement defaults spheroid measurement. To use the faster less accurate sphere use ST_Length(gg,false);

✓ This method implements the OpenGIS 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

Availability: 1.5.0 geography support was introduced in 1.5.

Geometry Examples

Return length in feet for line string. Note this is in feet because 2249 is Mass State Plane Feet

```
SELECT ST_Length(ST_GeomFromText('LINESTRING(743238 2967416,743238 2967450,743265 2967450,
743265.625 2967416,743238 2967416)',2249));
st_length
---------
 122.630744000095


--Transforming WGS 84 linestring to Massachusetts state plane meters
SELECT ST_Length(
  ST_Transform(
    ST_GeomFromEWKT('SRID=4326;LINESTRING(-72.1260 42.45, -72.1240 42.45666, -72.123  ↩
        42.1546)'),
    26986
  )
);
st_length
---------
34309.4563576191
```

Geography Examples

Return length of WGS 84 geography line

```
      -- default calculation is using a sphere rather than spheroid
SELECT ST_Length(the_geog) As length_spheroid,  ST_Length(the_geog,false) As length_sphere
FROM (SELECT ST_GeographyFromText(
'SRID=4326;LINESTRING(-72.1260 42.45, -72.1240 42.45666, -72.123 42.1546)') As the_geog)
 As foo;
 length_spheroid  |  length_sphere
------------------+------------------
 34310.5703627305 | 34346.2060960742
(1 row)
```

See Also

ST_GeographyFromText, ST_GeomFromEWKT, ST_Length_Spheroid, ST_Perimeter, ST_Transform

7.8.23 ST_Length2D

ST_Length2D — Returns the 2-dimensional length of the geometry if it is a linestring or multi-linestring. This is an alias for `ST_Length`

Synopsis

float **ST_Length2D**(geometry a_2dlinestring);

Description

Returns the 2-dimensional length of the geometry if it is a linestring or multi-linestring. This is an alias for `ST_Length`

See Also

ST_Length, ST_Length3D

7.8.24 ST_Length3D

ST_Length3D — Returns the 3-dimensional or 2-dimensional length of the geometry if it is a linestring or multi-linestring.

Synopsis

float **ST_Length3D**(geometry a_3dlinestring);

Description

Returns the 3-dimensional or 2-dimensional length of the geometry if it is a linestring or multi-linestring. For 2-d lines it will just return the 2-d length (same as ST_Length and ST_Length2D)

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

Examples

Return length in feet for a 3D cable. Note this is in feet because 2249 is Mass State Plane Feet

```
SELECT ST_Length3D(ST_GeomFromText('LINESTRING(743238 2967416 1,743238 2967450 1,743265  ↩
    2967450 3,
743265.625 2967416 3,743238 2967416 3)',2249));
st_length3d
-----------
122.704716741457
```

See Also

ST_Length, ST_Length2D

7.8.25 ST_Length_Spheroid

ST_Length_Spheroid — Calculates the 2D or 3D length of a linestring/multilinestring on an ellipsoid. This is useful if the coordinates of the geometry are in longitude/latitude and a length is desired without reprojection.

Synopsis

float **ST_Length_Spheroid**(geometry a_linestring, spheroid a_spheroid);

Description

Calculates the length of a geometry on an ellipsoid. This is useful if the coordinates of the geometry are in longitude/latitude and a length is desired without reprojection. The ellipsoid is a separate database type and can be constructed as follows:

```
SPHEROID[<NAME>,<SEMI-MAJOR
  AXIS>,<INVERSE FLATTENING>]
```

```
SPHEROID["GRS_1980",6378137,298.257222101]
```

Note
Will return 0 for anything that is not a MULTILINESTRING or LINESTRING

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

Examples

```
SELECT ST_Length_Spheroid( geometry_column,
        'SPHEROID["GRS_1980",6378137,298.257222101]' )
        FROM geometry_table;

SELECT ST_Length_Spheroid( the_geom, sph_m ) As tot_len,
ST_Length_Spheroid(ST_GeometryN(the_geom,1), sph_m) As len_line1,
ST_Length_Spheroid(ST_GeometryN(the_geom,2), sph_m) As len_line2
        FROM (SELECT ST_GeomFromText('MULTILINESTRING((-118.584 38.374,-118.583 38.5),
  (-71.05957 42.3589 , -71.061 43))') As the_geom,
CAST('SPHEROID["GRS_1980",6378137,298.257222101]' As spheroid) As sph_m)  as foo;
  tot_len      |    len_line1     |    len_line2
------------------+------------------+------------------
 85204.5207562955 | 13986.8725229309 | 71217.6482333646

 --3D
SELECT ST_Length_Spheroid( the_geom, sph_m ) As tot_len,
ST_Length_Spheroid(ST_GeometryN(the_geom,1), sph_m) As len_line1,
ST_Length_Spheroid(ST_GeometryN(the_geom,2), sph_m) As len_line2
        FROM (SELECT ST_GeomFromEWKT('MULTILINESTRING((-118.584 38.374 20,-118.583 38.5 30) ←↩
    ,
  (-71.05957 42.3589 75, -71.061 43 90))') As the_geom,
CAST('SPHEROID["GRS_1980",6378137,298.257222101]' As spheroid) As sph_m)  as foo;

   tot_len      |    len_line1    |    len_line2
------------------+-----------------+------------------
 85204.5259107402 | 13986.876097711 | 71217.6498130292
```

See Also

ST_GeometryN, ST_Length, ST_Length3D_Spheroid

7.8.26 ST_Length2D_Spheroid

ST_Length2D_Spheroid — Calculates the 2D length of a linestring/multilinestring on an ellipsoid. This is useful if the coordinates of the geometry are in longitude/latitude and a length is desired without reprojection.

Synopsis

float **ST_Length2D_Spheroid**(geometry a_linestring, spheroid a_spheroid);

Description

Calculates the 2D length of a geometry on an ellipsoid. This is useful if the coordinates of the geometry are in longitude/latitude and a length is desired without reprojection. The ellipsoid is a separate database type and can be constructed as follows:

```
SPHEROID[<NAME>,<SEMI-MAJOR
  AXIS>,<INVERSE FLATTENING>]
```

```
SPHEROID["GRS_1980",6378137,298.257222101]
```

Note
Will return 0 for anything that is not a MULTILINESTRING or LINESTRING

Note
This is much like ST_Length_Spheroid and ST_Length3D_Spheroid except it will throw away the Z coordinate in calculations.

Examples

```
SELECT ST_Length2D_Spheroid( geometry_column,
        ’SPHEROID["GRS_1980",6378137,298.257222101]’ )
        FROM geometry_table;

SELECT ST_Length2D_Spheroid( the_geom, sph_m ) As tot_len,
ST_Length2D_Spheroid(ST_GeometryN(the_geom,1), sph_m) As len_line1,
ST_Length2D_Spheroid(ST_GeometryN(the_geom,2), sph_m) As len_line2
        FROM (SELECT ST_GeomFromText(’MULTILINESTRING((-118.584 38.374,-118.583 38.5),
  (-71.05957 42.3589 , -71.061 43))’) As the_geom,
CAST(’SPHEROID["GRS_1980",6378137,298.257222101]’ As spheroid) As sph_m)  as foo;
  tot_len      |    len_line1     |    len_line2
------------------+------------------+------------------
 85204.5207562955 | 13986.8725229309 | 71217.6482333646

 --3D Observe same answer
SELECT ST_Length2D_Spheroid( the_geom, sph_m ) As tot_len,
ST_Length2D_Spheroid(ST_GeometryN(the_geom,1), sph_m) As len_line1,
ST_Length2D_Spheroid(ST_GeometryN(the_geom,2), sph_m) As len_line2
```

```
        FROM (SELECT ST_GeomFromEWKT('MULTILINESTRING((-118.584 38.374 20,-118.583 38.5 30) ↩
           ,
  (-71.05957 42.3589 75, -71.061 43 90))') As the_geom,
CAST('SPHEROID["GRS_1980",6378137,298.257222101]' As spheroid) As sph_m)  as foo;

  tot_len       |    len_line1     |    len_line2
------------------+------------------+------------------
 85204.5207562955 | 13986.8725229309 | 71217.6482333646
```

See Also

ST_GeometryN, ST_Length_Spheroid, ST_Length3D_Spheroid

7.8.27 ST_Length3D_Spheroid

ST_Length3D_Spheroid — Calculates the length of a geometry on an ellipsoid, taking the elevation into account. This is just an alias for ST_Length_Spheroid.

Synopsis

float **ST_Length3D_Spheroid**(geometry a_linestring, spheroid a_spheroid);

Description

Calculates the length of a geometry on an ellipsoid, taking the elevation into account. This is just an alias for ST_Length_Spheroid.

Note
Will return 0 for anything that is not a MULTILINESTRING or LINESTRING

Note
This functionis just an alias for ST_Length_Spheroid.

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

Examples

```
See ST_Length_Spheroid
```

See Also

ST_GeometryN, ST_Length, ST_Length_Spheroid

7.8.28 ST_LongestLine

ST_LongestLine — Returns the 2-dimensional longest line points of two geometries. The function will only return the first longest line if more than one, that the function finds. The line returned will always start in g1 and end in g2. The length of the line this function returns will always be the same as st_maxdistance returns for g1 and g2.

Synopsis

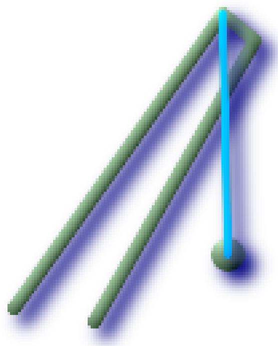
geometry **ST_LongestLine**(geometry g1, geometry g2);

Description

Returns the 2-dimensional longest line between the points of two geometries.

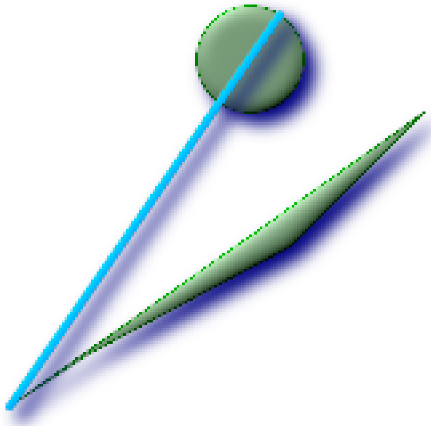
Availability: 1.5.0

Examples



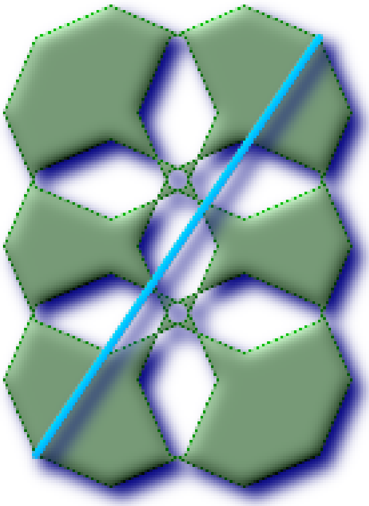
Longest line between point and line

```
SELECT ST_AsText(
        ST_LongestLine('POINT(100 100)':: ↩
    geometry,
                'LINESTRING (20 80, 98  ↩
    190, 110 180, 50 75 )'::geometry)
        ) As lline;


   lline
-----------------
LINESTRING(100 100,98 190)
```



longest line between polygon and polygon

```
SELECT ST_AsText(
        ST_LongestLine(
                ST_GeomFromText('POLYGON ↩
    ((175 150, 20 40,
                        50 60, 125 100,  ↩
    175 150))'),
                ST_Buffer(ST_GeomFromText ↩
    ('POINT(110 170)'), 20)
                )
        ) As llinewkt;

   lline
-----------------
LINESTRING(20 40,121.111404660392  ↩
    186.629392246051)
```



longest straight distance to travel from one part of an elegant city to the other Note the max distance = to the length of the line.

```
SELECT ST_AsText(ST_LongestLine(c.the_geom, c.the_geom)) As llinewkt,
        ST_MaxDistance(c.the_geom,c.the_geom) As max_dist,
        ST_Length(ST_LongestLine(c.the_geom, c.the_geom)) As lenll
FROM (SELECT ST_BuildArea(ST_Collect(the_geom)) As the_geom
        FROM (SELECT ST_Translate(ST_SnapToGrid(ST_Buffer(ST_Point(50 ,generate_series ↩
    (50,190, 50)
                        ),40, 'quad_segs=2'),1), x, 0)  As the_geom
                        FROM generate_series(1,100,50) As x)  AS foo
) As c;

          llinewkt           |      max_dist      |      lenll
---------------------------+------------------+------------------
 LINESTRING(23 22,129 178) | 188.605408193933 | 188.605408193933
```

See Also

ST_MaxDistance, ST_ShortestLine, ST_LongestLine

7.8.29 ST_OrderingEquals

ST_OrderingEquals — Returns true if the given geometries represent the same geometry and points are in the same directional order.

Synopsis

boolean **ST_OrderingEquals**(geometry A, geometry B);

Description

ST_OrderingEquals compares two geometries and 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

7.8.30 ST_Overlaps

ST_Overlaps — Returns TRUE if the Geometries share space, are of the same dimension, but are not completely contained by each other.

Synopsis

boolean **ST_Overlaps**(geometry A, geometry B);

Description

Returns TRUE if the Geometries "spatially overlap". By that we mean they intersect, but one does not completely contain another.

Performed by the GEOS module

Note
Do not call with a GeometryCollection as an argument

This function call will automatically include a bounding box comparison that will make use of any indexes that are available on the geometries. To avoid index use, use the function _ST_Overlaps.

NOTE: this is the "allowable" version that returns a boolean, not an integer.

This method implements the OpenGIS 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

```
--a point on a line is contained by the line and is of a lower dimension, and therefore ↩
    does not overlap the line
      nor crosses

SELECT ST_Overlaps(a,b) As a_overlap_b,
  ST_Crosses(a,b) As a_crosses_b,
    ST_Intersects(a, b) As a_intersects_b, ST_Contains(b,a) As b_contains_a
FROM (SELECT ST_GeomFromText('POINT(1 0.5)') As a, ST_GeomFromText('LINESTRING(1 0, 1 1, 3 ↩
    5)')  As b)
  As foo

a_overlap_b | a_crosses_b | a_intersects_b | b_contains_a
------------+-------------+----------------+--------------
f           | f           | t              | t

--a line that is partly contained by circle, but not fully is defined as intersecting and ↩
    crossing,
-- but since of different dimension it does not overlap
SELECT ST_Overlaps(a,b) As a_overlap_b, ST_Crosses(a,b) As a_crosses_b,
  ST_Intersects(a, b) As a_intersects_b,
  ST_Contains(a,b) As a_contains_b
FROM (SELECT ST_Buffer(ST_GeomFromText('POINT(1 0.5)'), 3)  As a, ST_GeomFromText(' ↩
    LINESTRING(1 0, 1 1, 3 5)')  As b)
  As foo;

 a_overlap_b | a_crosses_b | a_intersects_b | a_contains_b
-------------+-------------+----------------+--------------
 f           | t           | t              | f

 -- a 2-dimensional bent hot dog (aka puffered line string) that intersects a circle,
 -- but is not fully contained by the circle is defined as overlapping since they are of ↩
     the same dimension,
--  but it does not cross, because the intersection of the 2 is of the same dimension
--  as the maximum dimension of the 2

SELECT ST_Overlaps(a,b) As a_overlap_b, ST_Crosses(a,b) As a_crosses_b, ST_Intersects(a, b) ↩
     As a_intersects_b,
ST_Contains(b,a) As b_contains_a,
ST_Dimension(a) As dim_a, ST_Dimension(b) as dim_b, ST_Dimension(ST_Intersection(a,b)) As  ↩
    dima_intersection_b
FROM (SELECT ST_Buffer(ST_GeomFromText('POINT(1 0.5)'), 3)  As a,
  ST_Buffer(ST_GeomFromText('LINESTRING(1 0, 1 1, 3 5)'),0.5)  As b)
  As foo;

 a_overlap_b | a_crosses_b | a_intersects_b | b_contains_a | dim_a | dim_b |  ↩
     dima_intersection_b
-------------+-------------+----------------+--------------+-------+-------+--------------------- ↩

 t           | f           | t              | f            |     2 |     2 |                   2
```

See Also

ST_Contains, ST_Crosses, ST_Dimension, ST_Intersects

7.8.31 ST_Perimeter

ST_Perimeter — Return the length measurement of the boundary of an ST_Surface or ST_MultiSurface value. (Polygon, Multipolygon)

Synopsis

float **ST_Perimeter**(geometry g1);

Description

Returns the 2D perimeter of the geometry if it is a ST_Surface, ST_MultiSurface (Polygon, Multipolygon). 0 is returned for non-areal geometries. For linestrings use ST_Length. Measurements are in the units of the spatial reference system of the geometry.

Currently this is an alias for ST_Perimeter2D, but this may change to support higher dimensions.

✓ This method implements the OpenGIS 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

Examples

Return perimeter in feet for polygon and multipolygon. Note this is in feet because 2249 is Mass State Plane Feet

```
SELECT ST_Perimeter(ST_GeomFromText('POLYGON((743238 2967416,743238 2967450,743265 2967450,
743265.625 2967416,743238 2967416))', 2249));
st_perimeter
---------
 122.630744000095
(1 row)

SELECT ST_Perimeter(ST_GeomFromText('MULTIPOLYGON(((763104.471273676 2949418.44119003,
763104.477769673 2949418.42538203,
763104.189609677 2949418.22343004,763104.471273676 2949418.44119003)),
((763104.471273676 2949418.44119003,763095.804579742 2949436.33850239,
763086.132105649 2949451.46730207,763078.452329651 2949462.11549407,
763075.354136904 2949466.17407812,763064.362142565 2949477.64291974,
763059.953961626 2949481.28983009,762994.637609571 2949532.04103014,
762990.568508415 2949535.06640477,762986.710889563 2949539.61421415,
763117.237897679 2949709.50493431,763235.236617789 2949617.95619822,
763287.718121842 2949562.20592617,763111.553321674 2949423.91664605,
763104.471273676 2949418.44119003)))', 2249));
st_perimeter
---------
 845.227713366825
(1 row)
```

See Also

ST_Length

7.8.32 ST_Perimeter2D

ST_Perimeter2D — Returns the 2-dimensional perimeter of the geometry, if it is a polygon or multi-polygon. This is currently an alias for ST_Perimeter.

Synopsis

float **ST_Perimeter2D**(geometry geomA);

Description

Returns the 2-dimensional perimeter of the geometry, if it is a polygon or multi-polygon.

Note

This is currently an alias for ST_Perimeter. In future versions ST_Perimeter may return the highest dimension perimeter for a geometry. This is still under consideration

See Also

ST_Perimeter

7.8.33 ST_Perimeter3D

ST_Perimeter3D — Returns the 3-dimensional perimeter of the geometry, if it is a polygon or multi-polygon.

Synopsis

float **ST_Perimeter3D**(geometry geomA);

Description

Returns the 3-dimensional perimeter of the geometry, if it is a polygon or multi-polygon. If the geometry is 2-dimensional, then the 2-dimensional perimeter is returned.

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

Examples

Perimeter of a slightly elevated polygon in the air in Massachusetts state plane feet

```
SELECT ST_Perimeter3D(the_geom), ST_Perimeter2d(the_geom), ST_Perimeter(the_geom) FROM
      (SELECT ST_GeomFromEWKT('SRID=2249;POLYGON((743238 2967416 2,743238 2967450 1,
743265.625 2967416 1,743238 2967416 2))') As the_geom) As foo;

  st_perimeter3d  |  st_perimeter2d  |   st_perimeter
------------------+------------------+------------------
 105.465793597674 | 105.432997272188 | 105.432997272188
```

See Also

ST_GeomFromEWKT, ST_Perimeter, ST_Perimeter2D

7.8.34 ST_PointOnSurface

ST_PointOnSurface — Returns a POINT guaranteed to lie on the surface.

Synopsis

geometry **ST_PointOnSurface**(geometry g1);

Description

Returns a POINT guaranteed to intersect a surface.

This method implements the OpenGIS 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. According to the specs, ST_PointOnSurface works for surface geometries (POLYGONs, MULTIPOLYGONS, CURVED POLYGONS). So PostGIS seems to be extending what the spec allows here. Most databases Oracle,DB II, ESRI SDE seem to only support this function for surfaces. SQL Server 2008 like PostGIS supports for all common geometries.

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

Examples

```
SELECT ST_AsText(ST_PointOnSurface('POINT(0 5)'::geometry));
 st_astext
------------
 POINT(0 5)
(1 row)

SELECT ST_AsText(ST_PointOnSurface('LINESTRING(0 5, 0 10)'::geometry));
 st_astext
------------
 POINT(0 5)
(1 row)

SELECT ST_AsText(ST_PointOnSurface('POLYGON((0 0, 0 5, 5 5, 5 0, 0 0))'::geometry));
   st_astext
----------------
 POINT(2.5 2.5)
(1 row)

SELECT ST_AsEWKT(ST_PointOnSurface(ST_GeomFromEWKT('LINESTRING(0 5 1, 0 0 1, 0 10 2)')));
   st_asewkt
----------------
 POINT(0 0 1)
(1 row)
```

See Also

ST_Centroid, ST_Point_Inside_Circle

7.8.35 ST_Relate

ST_Relate — Returns true if this Geometry is spatially related to anotherGeometry, by testing for intersections between the Interior, Boundary and Exterior of the two geometries as specified by the values in the intersectionMatrixPattern. If no intersectionMatrixPattern is passed in, then returns the maximum intersectionMatrixPattern that relates the 2 geometries.

Synopsis

boolean **ST_Relate**(geometry geomA, geometry geomB, text intersectionMatrixPattern);
text **ST_Relate**(geometry geomA, geometry geomB);

Description

Version 1: Takes geomA, geomB, intersectionMatrix and Returns 1 (TRUE) if this Geometry is spatially related to anotherGeometry, by testing for intersections between the Interior, Boundary and Exterior of the two geometries as specified by the values in the intersectionMatrixPattern.

This is especially useful for testing compound checks of intersection, crosses, etc in one step.

Do not call with a GeometryCollection as an argument

Note
This is the "allowable" version that returns a boolean, not an integer. This is defined in OGC spec

Note
This DOES NOT automagically include an index call. The reason for that is some relationships are anti e.g. Disjoint. If you are using a relationship pattern that requires intersection, then include the && index call.

Version 2: Takes geomA and geomB and returns the Section 4.3.6

Note
Do not call with a GeometryCollection as an argument

not in OGC spec, but implied. see s2.1.13.2

Both Performed by the GEOS module

✓ This method implements the OpenGIS 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

Examples

```
--Find all compounds that intersect and not touch a poly (interior intersects)
SELECT l.* , b.name As poly_name
FROM polys As b
  INNER JOIN compounds As l
  ON (p.the_geom && b.the_geom
  AND ST_Relate(l.the_geom, b.the_geom,'T********'));

SELECT ST_Relate(ST_GeometryFromText('POINT(1 2)'), ST_Buffer(ST_GeometryFromText('POINT(1  ↩
    2)'),2));
 st_relate
-----------
 0FFFFF212

SELECT ST_Relate(ST_GeometryFromText('LINESTRING(1 2, 3 4)'), ST_GeometryFromText(' ↩
    LINESTRING(5 6, 7 8)'));
 st_relate
-----------
 FF1FF0102


SELECT ST_Relate(ST_GeometryFromText('POINT(1 2)'), ST_Buffer(ST_GeometryFromText('POINT(1  ↩
    2)'),2), '0FFFFF212');
 st_relate
-----------
 t

SELECT ST_Relate(ST_GeometryFromText('POINT(1 2)'), ST_Buffer(ST_GeometryFromText('POINT(1  ↩
    2)'),2), '*FF*FF212');
 st_relate
-----------
 t
```

See Also

ST_Crosses, Section 4.3.6, ST_Disjoint, ST_Intersects, ST_Touches

7.8.36 ST_ShortestLine

ST_ShortestLine — Returns the 2-dimensional shortest line between two geometries

Synopsis

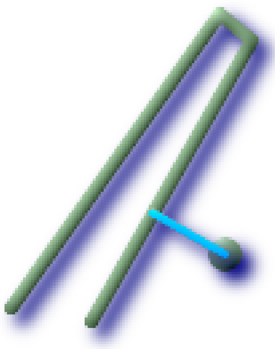
geometry **ST_ShortestLine**(geometry g1, geometry g2);

Description

Returns the 2-dimensional shortest line between two geometries. The function will only return the first shortest line if more than one, that the function finds. If g1 and g2 intersects in just one point the function will return a line with both start and end in that intersection-point. If g1 and g2 are intersecting with more than one point the function will return a line with start and end in the same point but it can be any of the intersecting points. The line returned will always start in g1 and end in g2. The length of the line this function returns will always be the same as st_distance returns for g1 and g2.

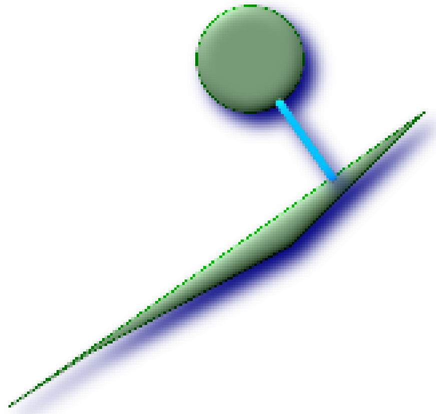
Availability: 1.5.0

Examples



Shortest line between point and linestring

```
SELECT ST_AsText(
        ST_ShortestLine('POINT(100 100) ↩
    '::geometry,
                   'LINESTRING (20 80, 98  ↩
    190, 110 180, 50 75 )'::geometry)
        ) As sline;


   sline
-----------------
LINESTRING(100 100,73.0769230769231  ↩
    115.384615384615)
```



shortest line between polygon and polygon

```
SELECT ST_AsText(
                ST_ShortestLine(
                        ST_GeomFromText(' ↩
    POLYGON((175 150, 20 40, 50 60, 125 100, 175 15
                        ST_Buffer( ↩
    ST_GeomFromText('POINT(110 170)'), 20)
                        )
                ) As slinewkt;

 LINESTRING(140.752120669087  ↩
    125.695053378061,121.111404660392 153.370607753
```

See Also

ST_ClosestPoint, ST_Distance, ST_LongestLine, ST_ShortestLine, ST_MaxDistance

7.8.37 ST_Touches

ST_Touches — Returns `TRUE` if the geometries have at least one point in common, but their interiors do not intersect.

Synopsis

boolean **ST_Touches**(geometry g1, geometry g2);

Description

Returns `TRUE` if the only points in common between *g1* and *g2* lie in the union of the boundaries of *g1* and *g2*. The `ST_Touches` relation applies to all Area/Area, Line/Line, Line/Area, Point/Area and Point/Line pairs of relationships, but *not* to the Point/Point pair.

In mathematical terms, this predicate is expressed as:

$$a.Touches(b) \Leftrightarrow (I(a) \cap I(b) = \emptyset) \wedge (a \cap b) \neq \emptyset$$

The allowable DE-9IM Intersection Matrices for the two geometries are:

- FT*******
- F**T*****
- F***T****

Important
Do not call with a GEOMETRYCOLLECTION as an argument

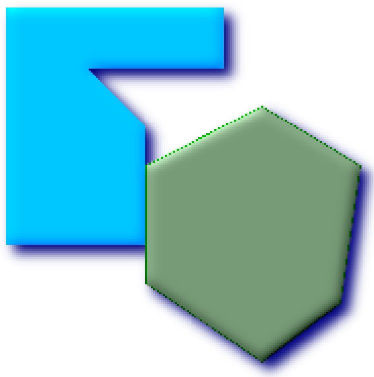
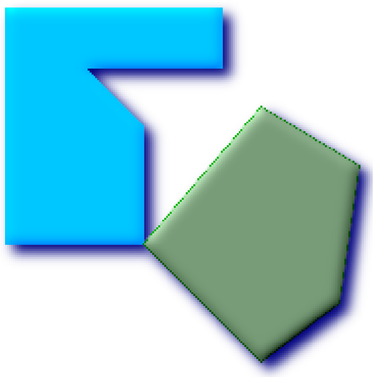
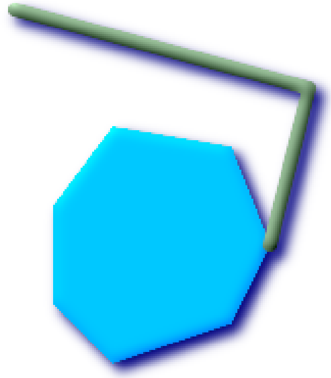
Note
This function call will automatically include a bounding box comparison that will make use of any indexes that are available on the geometries. To avoid using an index, use _ST_Touches instead.

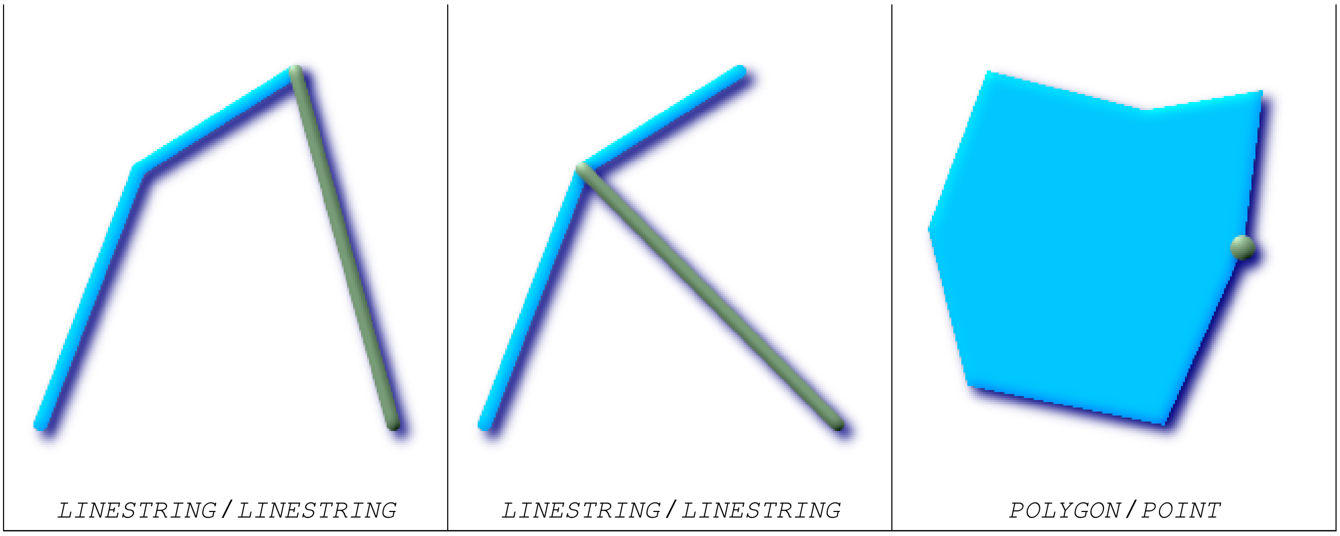
This method implements the OpenGIS Simple Features Implementation Specification for SQL 1.1. s2.1.1.2 // s2.1.13.3

This method implements the SQL/MM specification. SQL-MM 3: 5.1.28

Examples

The ST_Touches predicate returns TRUE in all the following illustrations.

 *POLYGON / POLYGON*	 *POLYGON / POLYGON*	 *POLYGON / LINESTRING*



LINESTRING / LINESTRING *LINESTRING / LINESTRING* *POLYGON / POINT*

```
SELECT ST_Touches('LINESTRING(0 0, 1 1, 0 2)'::geometry, 'POINT(1 1)'::geometry);
 st_touches
------------
 f
(1 row)

SELECT ST_Touches('LINESTRING(0 0, 1 1, 0 2)'::geometry, 'POINT(0 2)'::geometry);
 st_touches
------------
 t
(1 row)
```

7.8.38 ST_Within

ST_Within — Returns true if the geometry A is completely inside geometry B

Synopsis

boolean **ST_Within**(geometry A, geometry B);

Description

Returns TRUE if geometry A is completely inside geometry B. For this function to make sense, the source geometries must both be of the same coordinate projection, having the same SRID. It is a given that if ST_Within(A,B) is true and ST_Within(B,A) is true, then the two geometries are considered spatially equal.

Performed by the GEOS module

Important
Do not call with a `GEOMETRYCOLLECTION` as an argument

Important
Do not use this function with invalid geometries. You will get unexpected results.

This function call will automatically include a bounding box comparison that will make use of any indexes that are available on the geometries. To avoid index use, use the function _ST_Within.

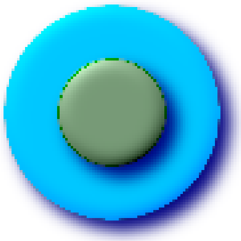
NOTE: this is the "allowable" version that returns a boolean, not an integer.

✓ This method implements the OpenGIS 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_Equals,ST_IsValid

7.9 Geometry Processing Functions

7.9.1 ST_Buffer

ST_Buffer — (T) For geometry: Returns a geometry that represents all points whose distance from this Geometry is less than or equal to distance. Calculations are in the Spatial Reference System of this Geometry. For geography: Uses a planar transform wrapper. Introduced in 1.5 support for different end cap and mitre settings to control shape. buffer_style options: quad_segs=#,endcap=round|flat|square,join=round|mitre|bevel,mitre_limit=#.#

Synopsis

geometry **ST_Buffer**(geometry g1, float radius_of_buffer);
geometry **ST_Buffer**(geometry g1, float radius_of_buffer, integer num_seg_quarter_circle);
geometry **ST_Buffer**(geometry g1, float radius_of_buffer, text buffer_style_parameters);
geography **ST_Buffer**(geography g1, float radius_of_buffer_in_meters);

Description

Returns a geometry/geography that represents all points whose distance from this Geometry/geography is less than or equal to distance.

Geometry: Calculations are in the Spatial Reference System of the geometry. Introduced in 1.5 support for different end cap and mitre settings to control shape.

Note
Negative radii: For polygons, a negative radius can be used, which will shrink the polygon rather than expanding it.

Note
Geography: For geography this is really a thin wrapper around the geometry implementation. It first determines the best SRID that fits the bounding box of the geography object (favoring UTM, Lambert Azimuthal Equal Area (LAEA) north/south pole, and falling back on mercator in worst case scenario) and then buffers in that planar spatial ref and retransforms back to WGS84 geography.

For geography this may not behave as expected if object is sufficiently large that it falls between two UTM zones or crosses the dateline

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. - requires GEOS >= 3.2 to take advantage of advanced geometry functionality.

The optional third parameter (currently only applies to geometry) can either specify number of segments used to approximate a quarter circle (integer case, defaults to 8) or a list of blank-separated key=value pairs (string case) to tweak operations as follows:

- 'quad_segs=#' : number of segments used to approximate a quarter circle (defaults to 8).
- 'endcap=round|flat|square' : endcap style (defaults to "round", needs GEOS-3.2 or higher for a different value). 'butt' is also accepted as a synonym for 'flat'.
- 'join=round|mitre|bevel' : join style (defaults to "round", needs GEOS-3.2 or higher for a different value). 'miter' is also accepted as a synonym for 'mitre'.
- 'mitre_limit=#.#' : mitre ratio limit (only affects mitred join style). 'miter_limit' is also accepted as a synonym for 'mitre_limit'.

Units of radius are measured in units of the spatial reference system.

The inputs can be POINTS, MULTIPOINTS, LINESTRINGS, MULTILINESTRINGS, POLYGONS, MULTIPOLYGONS, and GeometryCollections.

Note
This function ignores the third dimension (z) and will always give a 2-d buffer even when presented with a 3d-geometry.

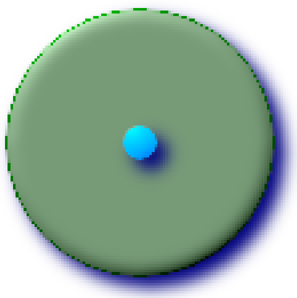
Performed by the GEOS module.

✓ This method implements the OpenGIS Simple Features Implementation Specification for SQL 1.1. s2.1.1.3

✓ This method implements the SQL/MM specification. SQL-MM 3: 5.1.17

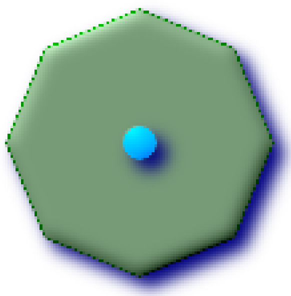
Note
People often make the mistake of using this function to try to do radius searches. Creating a buffer to to a radius search is slow and pointless. Use ST_DWithin instead.

Examples



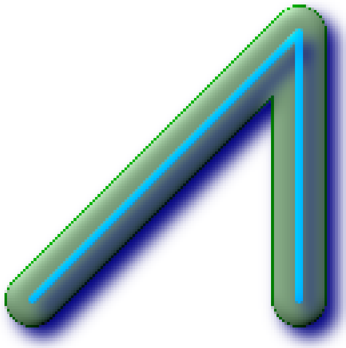
quad_segs=8 (default)

```
SELECT ST_Buffer(
 ST_GeomFromText('POINT(100 90)'),
 50, 'quad_segs=8');
```



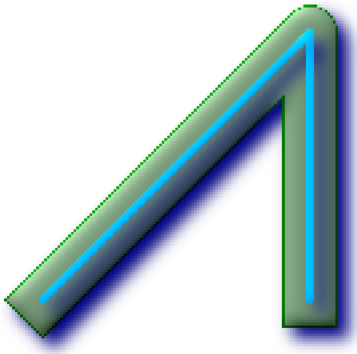
quad_segs=2 (lame)

```
SELECT ST_Buffer(
 ST_GeomFromText('POINT(100 90)'),
 50, 'quad_segs=2');
```



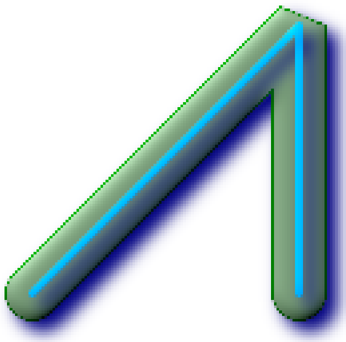
endcap=round join=round (default)

```
SELECT ST_Buffer(
 ST_GeomFromText(
  'LINESTRING(50 50,150 150,150 50)'
 ), 10, 'endcap=round join=round');
```



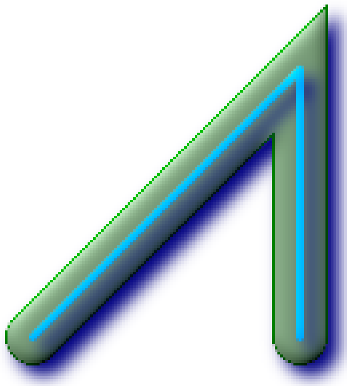
endcap=square

```
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');
```

```
--A buffered point approximates a circle
```

```
-- A buffered point forcing approximation of (see diagram)
-- 2 points per circle is poly with 8 sides (see diagram)
SELECT ST_NPoints(ST_Buffer(ST_GeomFromText('POINT(100 90)'), 50)) As  ↩
    promisingcircle_pcount,
ST_NPoints(ST_Buffer(ST_GeomFromText('POINT(100 90)'), 50, 2)) As lamecircle_pcount;

promisingcircle_pcount | lamecircle_pcount
------------------------+-------------------
       33 |                 9

--A lighter but lamer circle
-- only 2 points per quarter circle is an octagon
--Below is a 100 meter octagon
-- 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_MakePoint(-71.063526, 42.35785),4269), 26986)
,100,2)) As octagon;
----------------------
POLYGON((236057.59057465 900908.759918696,236028.301252769 900838.049240578,235
957.59057465 900808.759918696,235886.879896532 900838.049240578,235857.59057465
900908.759918696,235886.879896532 900979.470596815,235957.59057465 901008.759918
696,236028.301252769 900979.470596815,236057.59057465 900908.759918696))
```

See Also

ST_Collect, ST_DWithin, ST_SetSRID, ST_Transform, ST_Union

7.9.2 ST_BuildArea

ST_BuildArea — Creates an areal geometry formed by the constituent linework of given geometry

Synopsis

geometry **ST_BuildArea**(geometry A);

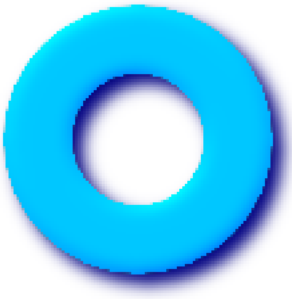
Description

Creates an areal geometry formed by the constituent linework of given geometry. The return type can be a Polygon or MultiPolygon, depending on input. If the input lineworks do not form polygons NULL is returned. The inputs can be LINESTRINGS, MULTILINESTRINGS, POLYGONS, MULTIPOLYGONS, and GeometryCollections.

This function will assume all inner geometries represent holes

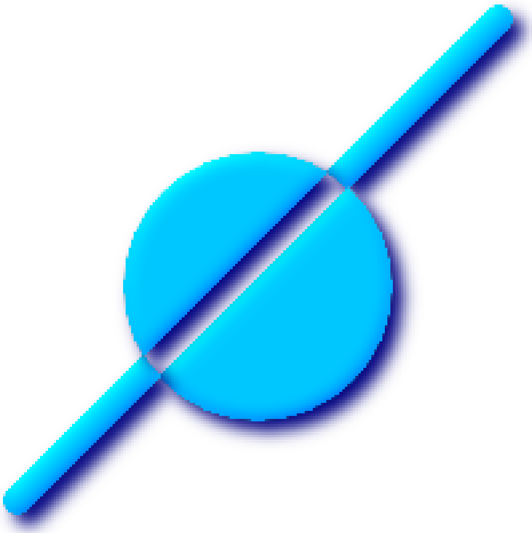
Availability: 1.1.0 - requires GEOS >= 2.1.0.

Examples



This will create a donut

```
SELECT ST_BuildArea(ST_Collect(smallc,bigc))
FROM (SELECT
        ST_Buffer(
          ST_GeomFromText('POINT(100 90)'), 25) As smallc,
        ST_Buffer(ST_GeomFromText('POINT(100 90)'), 50) As bigc) As foo;
```



This will create a gaping hole inside the circle with prongs sticking out

```
SELECT ST_BuildArea(ST_Collect(line,circle))
FROM (SELECT
        ST_Buffer(
                ST_MakeLine(ST_MakePoint(10, 10),ST_MakePoint(190, 190)),
                                5)  As line,
        ST_Buffer(ST_GeomFromText('POINT(100 90)'), 50) As circle) As foo;

--this creates the same gaping hole
--but using linestrings instead of polygons
SELECT ST_BuildArea(
        ST_Collect(ST_ExteriorRing(line),ST_ExteriorRing(circle))
        )
FROM (SELECT ST_Buffer(
        ST_MakeLine(ST_MakePoint(10, 10),ST_MakePoint(190, 190))
                ,5)  As line,
        ST_Buffer(ST_GeomFromText('POINT(100 90)'), 50) As circle) As foo;
```

See Also

ST_BdPolyFromText, ST_BdMPolyFromTextwrappers to this function with standard OGC interface

7.9.3 ST_Collect

ST_Collect — Return a specified ST_Geometry value from a collection of other geometries.

Synopsis

geometry **ST_Collect**(geometry set g1field);
geometry **ST_Collect**(geometry g1, geometry g2);
geometry **ST_Collect**(geometry[] g1_array);

Description

Output type can be a MULTI* or a GEOMETRYCOLLECTION. Comes in 2 variants. Variant 1 collects 2 geometries. Variant 2 is an aggregate function that takes a set of geometries and collects them into a single ST_Geometry.

Aggregate version: This function returns a GEOMETRYCOLLECTION or a MULTI object from a set of geometries. The ST_Collect() 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. For example, "SELECT ST_Collect(GEOM) FROM GEOMTABLE GROUP BY ATTRCOLUMN" will return a separate GEOMETRYCOLLECTION for each distinct value of ATTRCOLUMN.

Non-Aggregate version: This function returns a geometry being a collection of two input geometries. Output type can be a MULTI* or a GEOMETRYCOLLECTION.

Note
ST_Collect and ST_Union are often interchangeable. ST_Collect is in general orders of magnitude faster than ST_Union because it does not try to dissolve boundaries or validate that a constructed MultiPolgon doesn't have overlapping regions. It merely rolls up single geometries into MULTI and MULTI or mixed geometry types into Geometry Collections. Unfortunately geometry collections are not well-supported by GIS tools. To prevent ST_Collect from returning a Geometry Collection when collecting MULTI geometries, one can use the below trick that utilizes ST_Dump to expand the MULTIs out to singles and then regroup them.

Availability: 1.4.0 - ST_Collect(geomarray) was introduced. ST_Collect was enhanced to handle more geometries faster.

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

✓ This method supports Circular Strings and Curves This method supports Circular Strings and Curves, but will never return a MULTICURVE or MULTI as one would expect and PostGIS does not currently support those.

Examples

Aggregate example

```
Thread ref: http://postgis.refractions.net/pipermail/postgis-users/2008-June/020331.html
SELECT stusps,
     ST_Multi(ST_Collect(f.the_geom)) as singlegeom
   FROM (SELECT stusps, (ST_Dump(the_geom)).geom As the_geom
        FROM
        somestatetable ) As f
GROUP BY stusps
```

Non-Aggregate example

```
Thread ref: http://postgis.refractions.net/pipermail/postgis-users/2008-June/020331.html
SELECT ST_AsText(ST_Collect(ST_GeomFromText('POINT(1 2)'),
  ST_GeomFromText('POINT(-2 3)') ));

st_astext
----------
MULTIPOINT(1 2,-2 3)

--Collect 2 d points
SELECT ST_AsText(ST_Collect(ST_GeomFromText('POINT(1 2)'),
    ST_GeomFromText('POINT(1 2)') ) );

st_astext
----------
MULTIPOINT(1 2,1 2)
```

```
--Collect 3d points
SELECT ST_AsEWKT(ST_Collect(ST_GeomFromEWKT('POINT(1 2 3)'),
    ST_GeomFromEWKT('POINT(1 2 4)') ) );

    st_asewkt
-------------------------
 MULTIPOINT(1 2 3,1 2 4)

 --Example with curves
SELECT ST_AsText(ST_Collect(ST_GeomFromText('CIRCULARSTRING(220268 150415,220227  ↩
    150505,220227 150406)'),
ST_GeomFromText('CIRCULARSTRING(220227 150406,2220227 150407,220227 150406)')));
                                st_astext
------------------------------------------------------------------------------------
 GEOMETRYCOLLECTION(CIRCULARSTRING(220268 150415,220227 150505,220227 150406),
 CIRCULARSTRING(220227 150406,2220227 150407,220227 150406))

--New ST_Collect array construct
SELECT ST_Collect(ARRAY(SELECT the_geom FROM sometable));

SELECT ST_AsText(ST_Collect(ARRAY[ST_GeomFromText('LINESTRING(1 2, 3 4)'),
      ST_GeomFromText('LINESTRING(3 4, 4 5)')])) As wktcollect;

--wkt collect --
MULTILINESTRING((1 2,3 4),(3 4,4 5))
```

See Also

ST_Dump, ST_Union

7.9.4 ST_ConvexHull

ST_ConvexHull — The convex hull of a geometry represents the minimum convex geometry that encloses all geometries within the set.

Synopsis

geometry **ST_ConvexHull**(geometry geomA);

Description

The convex hull of a geometry represents the minimum convex geometry that encloses all geometries within the set.

One can think of the convex hull as the geometry you get by wrapping an elastic band around a set of geometries. This is different from a concave hull (not currently supported) which is analogous to shrink-wrapping your geometries.

It is usually used with MULTI and Geometry Collections. Although it is not an aggregate - you can use it in conjunction with ST_Collect to get the convex hull of a set of points. ST_ConvexHull(ST_Collect(somepointfield)).

It is often used to determine an affected area based on a set of point observations.

Performed by the GEOS module

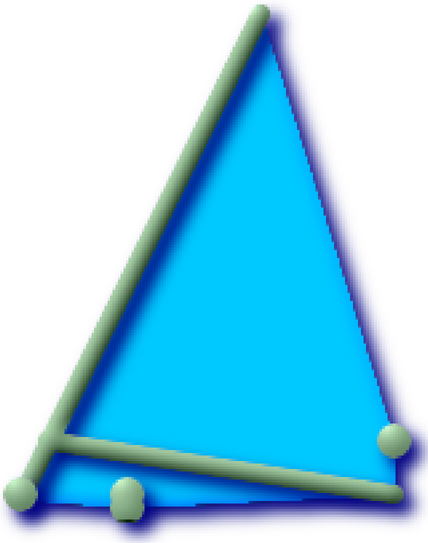
✓ This method implements the OpenGIS Simple Features Implementation Specification for SQL 1.1. s2.1.1.3

✓ This method implements the SQL/MM specification. SQL-MM 3: 5.1.16

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

Examples

```
--Get estimate of infected area based on point observations
SELECT d.disease_type,
  ST_ConvexHull(ST_Collect(d.the_geom)) As the_geom
  FROM disease_obs As d
  GROUP BY d.disease_type;
```



Convex Hull of a MultiLinestring and a MultiPoint seen together with the MultiLinestring and MultiPoint

```
SELECT ST_AsText(ST_ConvexHull(
  ST_Collect(
    ST_GeomFromText('MULTILINESTRING((100 190,10 8),(150 10, 20 30))'),
      ST_GeomFromText('MULTIPOINT(50 5, 150 30, 50 10, 10 10)')
      )) );
---st_astext--
POLYGON((50 5,10 8,10 10,100 190,150 30,150 10,50 5))
```

See Also

ST_Collect, ST_MinimumBoundingCircle

7.9.5 ST_CurveToLine

ST_CurveToLine — Converts a CIRCULARSTRING/CURVEDPOLYGON to a LINESTRING/POLYGON

Synopsis

geometry **ST_CurveToLine**(geometry curveGeom);
geometry **ST_CurveToLine**(geometry curveGeom, integer segments_per_qtr_circle);

Description

Converst a CIRCULAR STRING to regular LINESTRING or CURVEPOLYGON to POLYGON. 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 default value of 32 segments per quarter circle

Availability: 1.2.2?

✓ This method implements the OpenGIS 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

Examples

```
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)
```

See Also

ST_LineToCurve

7.9.6 ST_Difference

ST_Difference — Returns a geometry that represents that part of geometry A that does not intersect with geometry B.

Synopsis

geometry **ST_Difference**(geometry geomA, geometry geomB);

Description

Returns a geometry that represents that part of geometry A that does not intersect with geometry B. One can think of this as GeometryA - ST_Intersection(A,B). If A is completely contained in B then an empty geometry collection is returned.

Note
Note - order matters. B - A will always return a portion of B

Performed by the GEOS module

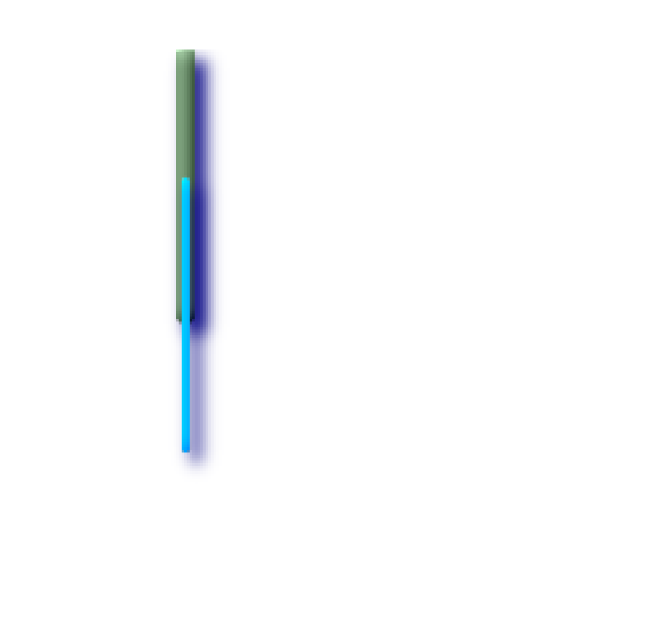
Note
Do not call with a GeometryCollection as an argument

✓ This method implements the OpenGIS Simple Features Implementation Specification for SQL 1.1. s2.1.1.3

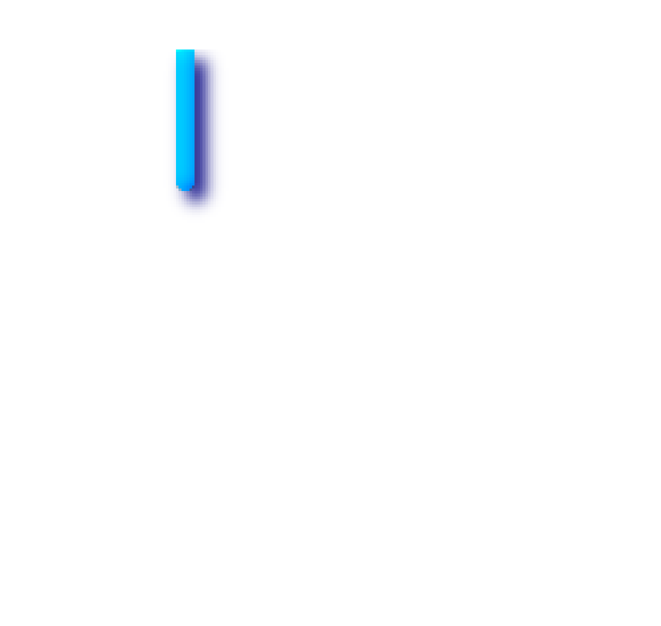
✓ This method implements the SQL/MM specification. SQL-MM 3: 5.1.20

✓ This function supports 3d and will not drop the z-index. However it seems to only consider x y when doing the difference and tacks back on the Z-Index

Examples



The original linestrings shown together.



The difference of the two linestrings

```
--Safe for 2d. This is same geometries as what is shown for st_symdifference
SELECT ST_AsText(
  ST_Difference(
      ST_GeomFromText('LINESTRING(50 100, 50 200)'),
      ST_GeomFromText('LINESTRING(50 50, 50 150)')
    )
  );

st_astext
---------
LINESTRING(50 150,50 200)
```

```
--When used in 3d doesn't quite do the right thing
SELECT ST_AsEWKT(ST_Difference(ST_GeomFromEWKT('MULTIPOINT(-118.58 38.38 5,-118.60 38.329  ↩
    6,-118.614 38.281 7)'), ST_GeomFromEWKT('POINT(-118.614 38.281 5)')));
st_asewkt
---------
MULTIPOINT(-118.6 38.329 6,-118.58 38.38 5)
```

See Also

ST_SymDifference

7.9.7 ST_Dump

ST_Dump — Returns a set of geometry_dump (geom,path) rows, that make up a geometry g1.

Synopsis

geometry_dump[]**ST_Dump**(geometry g1);

Description

This is a set-returning function (SRF). It returns a set of geometry_dump rows, formed by a geometry (geom) and an array of integers (path). When the input geometry is a simple type (POINT,LINESTRING,POLYGON) a single record will be returned with an empty path array and the input geometry as geom. When the input geometry is a collection or multi it will return a record for each of the collection components, and the path will express the position of the component inside the collection.

ST_Dump is useful for expanding geometries. It is the reverse of a GROUP BY in that it creates new rows. For example it can be use to expand MULTIPOLYGONS into POLYGONS.

Availability: PostGIS 1.0.0RC1. Requires PostgreSQL 7.3 or higher.

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

Examples

```
SELECT sometable.field1, sometable.field1,
      (ST_Dump(sometable.the_geom)).geom AS the_geom
FROM sometable;

--Break a compound curve into its constituent linestrings and circularstrings
SELECT ST_AsEWKT(a.geom), ST_HasArc(a.geom)
  FROM ( SELECT (ST_Dump(p_geom)).geom AS geom
         FROM (SELECT ST_GeomFromEWKT('COMPOUNDCURVE(CIRCULARSTRING(0 0, 1 1, 1 0),(1 0, 0  ↩
             1))') AS p_geom) AS b
        ) AS a;
          st_asewkt           | st_hasarc
-----------------------------+----------
 CIRCULARSTRING(0 0,1 1,1 0) | t
 LINESTRING(1 0,0 1)         | f
(2 rows)
```

See Also

geometry_dump, Section 8.4, ST_Collect, ST_Collect, ST_GeometryN

7.9.8 ST_DumpPoints

ST_DumpPoints — Returns a set of geometry_dump (geom,path) rows of all points that make up a geometry.

Synopsis

geometry_dump[]**ST_DumpPoints**(geometry geom);

Description

This set-returning function (SRF) returns a set of `geometry_dump` rows formed by a geometry (`geom`) and an array of integers (`path`).

The *`geom`* component of `geometry_dump` are all the `POINT`s that make up the supplied geometry

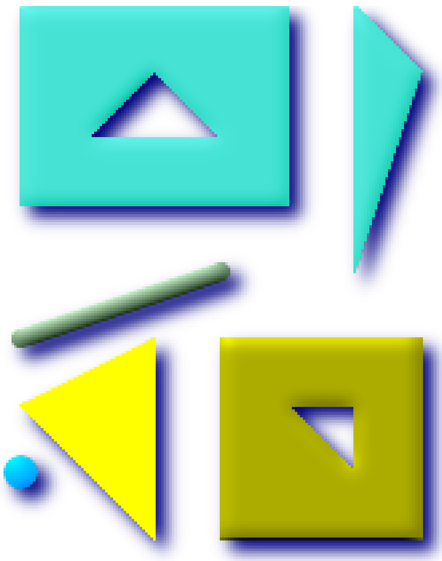
The *`path`* component of `geometry_dump` (an `integer[]`) is an index reference enumerating the `POINT`s of the supplied geometry. For example, if a `LINESTRING` is supplied, a path of `{i}` is returned where `i` is the `nth` coordinate in the `LINESTRING`. If a `POLYGON` is supplied, a path of `{i,j}` is returned where `i` is the outer ring followed by the inner rings and `j` enumerates the `POINT`s.

Availability: 1.5.0

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

Throws an error if curved geometries or other unsupported geometries are used.

Examples



```
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)
```

See Also

geometry_dump, Section 8.4, ST_Dump, ST_DumpRings

7.9.9 ST_DumpRings

ST_DumpRings — Returns a set of `geometry_dump` rows, representing the exterior and interior rings of a polygon.

Synopsis

geometry_dump[] **ST_DumpRings**(geometry a_polygon);

Description

This is a set-returning function (SRF). It returns a set of `geometry_dump` rows, defined as an `integer[]` and a `geometry`, aliased "path" and "geom" respectively. The "path" field holds the polygon ring index containing a single integer: 0 for the shell, >0 for holes. The "geom" field contains the corresponding ring as a polygon.

Availability: PostGIS 1.1.3. Requires PostgreSQL 7.3 or higher.

Note

This only works for POLYGON geometries. It will not work for MULTIPOLYGONS

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

Examples

```
SELECT sometable.field1, sometable.field1,
    (ST_DumpRings(sometable.the_geom)).geom As the_geom
FROM sometableOfpolys;

SELECT ST_AsEWKT(geom) As the_geom, path
  FROM ST_DumpRings(
    ST_GeomFromEWKT('POLYGON((-8149064 5133092 1,-8149064 5132986 1,-8148996 5132839  ↩
        1,-8148972 5132767 1,-8148958 5132508 1,-8148941 5132466 1,-8148924 5132394 1,
    -8148903 5132210 1,-8148930 5131967 1,-8148992 5131978 1,-8149237 5132093 1,-8149404  ↩
        5132211 1,-8149647 5132310 1,-8149757 5132394 1,
    -8150305 5132788 1,-8149064 5133092 1),
    (-8149362 5132394 1,-8149446 5132501 1,-8149548 5132597 1,-8149695 5132675 1,-8149362  ↩
        5132394 1))')
    )  as foo;
 path |                                            the_geom
----------------------------------------------------------------------------------------------------

  {0} | POLYGON((-8149064 5133092 1,-8149064 5132986 1,-8148996 5132839 1,-8148972 5132767  ↩
      1,-8148958 5132508 1,
    |          -8148941 5132466 1,-8148924 5132394 1,
    |          -8148903 5132210 1,-8148930 5131967 1,
    |          -8148992 5131978 1,-8149237 5132093 1,
    |          -8149404 5132211 1,-8149647 5132310 1,-8149757 5132394 1,-8150305 5132788  ↩
        1,-8149064 5133092 1))
  {1} | POLYGON((-8149362 5132394 1,-8149446 5132501 1,
    |          -8149548 5132597 1,-8149695 5132675 1,-8149362 5132394 1))
```

See Also

geometry_dump, Section 8.4, ST_Dump, ST_ExteriorRing, ST_InteriorRingN

7.9.10 ST_Intersection

ST_Intersection — (T) Returns a geometry that represents the shared portion of geomA and geomB. The geography implementation does a transform to geometry to do the intersection and then transform back to WGS84.

Synopsis

geometry **ST_Intersection**(geometry geomA , geometry geomB);
geography **ST_Intersection**(geography geogA , geography geogB);

Description

Returns a geometry that represents the point set intersection of the Geometries.

In other words - that portion of geometry A and geometry B that is shared between the two geometries.

If the geometries do not share any space (are disjoint), then an empty geometry collection is returned.

ST_Intersection in conjunction with ST_Intersects is very useful for clipping geometries such as in bounding box, buffer, region queries where you only want to return that portion of a geometry that sits in a country or region of interest.

Note
Geography: For geography this is really 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.

Important
Do not call with a `GEOMETRYCOLLECTION` as an argument

Performed by the GEOS module

Availability: 1.5 support for geography data type was introduced.

✓ This method implements the OpenGIS 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

Examples

```
SELECT ST_AsText(ST_Intersection(’POINT(0 0)’::geometry, ’LINESTRING ( 2 0, 0 2 )’::geometry));
 st_astext
---------------
GEOMETRYCOLLECTION EMPTY
(1 row)
SELECT ST_AsText(ST_Intersection(’POINT(0 0)’::geometry, ’LINESTRING ( 0 0, 0 2 )’::geometry));
 st_astext
---------------
POINT(0 0)
(1 row)

---Clip all lines (trails) by country (here we assume country geom are POLYGON or MULTIPOLYGONS)
-- NOTE: we are only keeping intersections that result in a LINESTRING or MULTILINESTRING because we don’t
-- care about trails that just share a point
-- the dump is needed to expand a geometry collection into individual single MULT* parts
-- the below is fairly generic and will work for polys, etc. by just changing the where clause
SELECT clipped.gid, clipped.f_name, clipped_geom
FROM (SELECT trails.gid, trails.f_name, (ST_Dump(ST_Intersection(country.the_geom, trails.the_geom))).geom As clipped_geom
FROM country
  INNER JOIN trails
  ON ST_Intersects(country.the_geom, trails.the_geom))  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.the_geom, poly.the_geom),
        0.0)
        ) As clipped_geom
FROM country
  INNER JOIN poly
  ON ST_Intersects(country.the_geom, poly.the_geom)
  WHERE Not ST_IsEmpty(ST_Buffer(ST_Intersection(country.the_geom, poly.the_geom),0.0));
```

See Also

ST_Difference, ST_Dimension, ST_Dump, ST_SymDifference, ST_Intersects, ST_Multi

7.9.11 ST_LineToCurve

ST_LineToCurve — Converts a LINESTRING/POLYGON to a CIRCULARSTRING, CURVED POLYGON

Synopsis

geometry **ST_LineToCurve**(geometry geomANoncircular);

Description

Converts plain LINESTRING/POLYGONS to CIRCULAR STRINGs and Curved Polygons. Note much fewer points are needed to describe the curved equivalent.

Availability: 1.2.2?

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

✓ This method supports Circular Strings and Curves

Examples

```
SELECT ST_AsText(ST_LineToCurve(foo.the_geom)) As curvedastext,ST_AsText(foo.the_geom) As  ↩
    non_curvedastext
  FROM (SELECT ST_Buffer('POINT(1 3)'::geometry, 3) As the_geom) As foo;

curvedatext                                    non_curvedastext
------------------------------------------------------------------|  ↩
    ----------------------------------------------------------------
CURVEPOLYGON(CIRCULARSTRING(4 3,3.12132034355964 0.878679656440359,  |  POLYGON((4  ↩
    3,3.94235584120969 2.41472903395162,3.77163859753386 1.85194970290473
1 0,-1.12132034355965 5.12132034355963,4 3))              | ,3.49440883690764  ↩
    1.33328930094119,3.12132034355964 0.878679656440359,
                                  |      2.66671069905881  ↩
                                     0.505591163092366,2.14805029709527 0.228361402466141,
                                  |      1.58527096604839 0.0576441587903094,1 0,
                                  |      0.414729033951621  ↩
                                     0.0576441587903077,-0.148050297095264  ↩
                                     0.228361402466137,
                                  |      -0.666710699058802  ↩
                                     0.505591163092361,-1.12132034355964 0.878679656440353,
                                  |      -1.49440883690763  ↩
                                     1.33328930094119,-1.77163859753386 1.85194970290472
                                  |       --ETC-- ,3.94235584120969 3.58527096604839,4 3))
--3D example
SELECT ST_AsEWKT(ST_LineToCurve(ST_GeomFromEWKT('LINESTRING(1 2 3, 3 4 8, 5 6 4, 7 8 4, 9  ↩
    10 4)')));

       st_asewkt
------------------------------------
 CIRCULARSTRING(1 2 3,5 6 4,9 10 4)
```

See Also

ST_CurveToLine

7.9.12 ST_MemUnion

ST_MemUnion — Same as ST_Union, only memory-friendly (uses less memory and more processor time).

Synopsis

geometry **ST_MemUnion**(geometry set geomfield);

Description

Some useful description here.

> **Note**
> Same as ST_Union, only memory-friendly (uses less memory and more processor time). This aggregate function works by unioning the geometries one at a time to previous result as opposed to ST_Union aggregate which first creates an array and then unions

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

Examples

```
See ST_Union
```

See Also

ST_Union

7.9.13 ST_MinimumBoundingCircle

ST_MinimumBoundingCircle — Returns the smallest circle polygon that can fully contain a geometry. Default uses 48 segments per quarter circle.

Synopsis

geometry **ST_MinimumBoundingCircle**(geometry geomA);
geometry **ST_MinimumBoundingCircle**(geometry geomA, integer num_segs_per_qt_circ);

Description

Returns the smallest circle polygon that can fully contain a geometry.

> **Note**
> The circle is approximated by a polygon with a default of 48 segments per quarter circle. This number can be increased with little performance penalty to obtain a more accurate result.

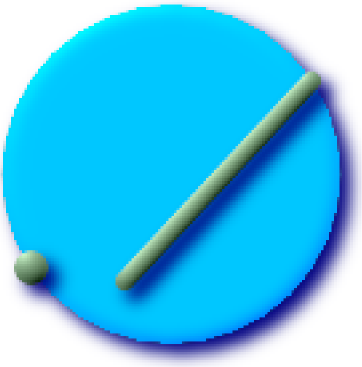
It is often used with MULTI and Geometry Collections. Although it is not an aggregate - you can use it in conjunction with ST_Collect to get the minimum bounding cirlce of a set of geometries. ST_MinimumBoundingCircle(ST_Collect(somepointfield)).

The ratio of the area of a polygon divided by the area of its Minimum Bounding Circle is often referred to as the Roeck test.

Availability: 1.4.0 - requires GEOS

Examples

```
SELECT d.disease_type,
  ST_MinimumBoundingCircle(ST_Collect(d.the_geom)) As the_geom
  FROM disease_obs As d
  GROUP BY d.disease_type;
```



Minimum bounding circle of a point and linestring. Using 8 segs to approximate a quarter circle

```
SELECT ST_AsText(ST_MinimumBoundingCircle(
    ST_Collect(
      ST_GeomFromEWKT('LINESTRING(55 75,125 150)'),
        ST_Point(20, 80)), 8
        )) As wktmbc;
wktmbc
-----------
POLYGON((135.59714732062 115,134.384753327498 102.690357210921,130.79416296937 ↩
    90.8537670908995,124.963360620072 79.9451031602111,117.116420743937 ↩
    70.3835792560632,107.554896839789 62.5366393799277,96.6462329091006 ↩
    56.70583703063,84.8096427890789 53.115246672502,72.5000000000001 ↩
    51.9028526793802,60.1903572109213 53.1152466725019,48.3537670908996 ↩
    56.7058370306299,37.4451031602112 62.5366393799276,27.8835792560632 ↩
    70.383579256063,20.0366393799278 79.9451031602109,14.20583703063  ↩
    90.8537670908993,10.615246672502 102.690357210921,9.40285267938019 115,10.6152466725019  ↩
    127.309642789079,14.2058370306299 139.1462329091,20.0366393799275  ↩
    150.054896839789,27.883579256063 159.616420743937,
37.4451031602108 167.463360620072,48.3537670908992 173.29416296937,60.190357210921  ↩
    176.884753327498,
72.4999999999998 178.09714732062,84.8096427890786 176.884753327498,96.6462329091003  ↩
    173.29416296937,107.554896839789 167.463360620072,
117.116420743937 159.616420743937,124.963360620072 150.054896839789,130.79416296937  ↩
    139.146232909101,134.384753327498 127.309642789079,135.59714732062 115))
```

See Also

ST_Collect, ST_ConvexHull

7.9.14 ST_Polygonize

ST_Polygonize — Aggregate. Creates a GeometryCollection containing possible polygons formed from the constituent linework of a set of geometries.

Synopsis

geometry **ST_Polygonize**(geometry set geomfield);
geometry **ST_Polygonize**(geometry[] geom_array);

Description

Creates a GeometryCollection containing possible polygons formed from the constituent linework of a set of geometries.

Note
Geometry Collections are often difficult to deal with with third party tools, so use ST_Polygonize in conjunction with ST_Dump to dump the polygons out into individual polygons.

Availability: 1.0.0RC1 - requires GEOS >= 2.1.0.

Examples: Polygonizing single linestrings

```
SELECT ST_AsEWKT(ST_Polygonize(the_geom_4269)) As geomtextrep
FROM (SELECT the_geom_4269 FROM ma.suffolk_edges ORDER BY tlid LIMIT 45) As foo;

geomtextrep
-------------------------------------
 SRID=4269;GEOMETRYCOLLECTION(POLYGON((-71.040878 42.285678,-71.040943 42.2856,-71.04096  ↩
     42.285752,-71.040878 42.285678)),
 POLYGON((-71.17166 42.353675,-71.172026 42.354044,-71.17239 42.354358,-71.171794  ↩
     42.354971,-71.170511 42.354855,
 -71.17112 42.354238,-71.17166 42.353675)))
(1 row)

--Use ST_Dump to dump out the polygonize geoms into individual polygons
SELECT ST_AsEWKT((ST_Dump(foofoo.polycoll)).geom) As geomtextrep
FROM (SELECT ST_Polygonize(the_geom_4269) As polycoll
  FROM (SELECT the_geom_4269 FROM ma.suffolk_edges
    ORDER BY tlid LIMIT 45) As foo) As foofoo;

geomtextrep
------------------------
 SRID=4269;POLYGON((-71.040878 42.285678,-71.040943 42.2856,-71.04096 42.285752,
-71.040878 42.285678))
 SRID=4269;POLYGON((-71.17166 42.353675,-71.172026 42.354044,-71.17239 42.354358
,-71.171794 42.354971,-71.170511 42.354855,-71.17112 42.354238,-71.17166 42.353675))
(2 rows)
```

See Also

ST_Dump

7.9.15 ST_Shift_Longitude

ST_Shift_Longitude — Reads every point/vertex in every component of every feature in a geometry, and if the longitude coordinate is <0, adds 360 to it. The result would be a 0-360 version of the data to be plotted in a 180 centric map

Synopsis

geometry **ST_Shift_Longitude**(geometry geomA);

Description

Reads every point/vertex in every component of every feature in a geometry, and if the longitude coordinate is <0, adds 360 to it. The result would be a 0-360 version of the data to be plotted in a 180 centric map

Note
This is only useful for data in long lat e.g. 4326 (WGS 84 long lat)

Pre-1.3.4 bug prevented this from working for MULTIPOINT. 1.3.4+ works with MULTIPOINT as well.

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

Examples

```
--3d points
SELECT ST_AsEWKT(ST_Shift_Longitude(ST_GeomFromEWKT('SRID=4326;POINT(-118.58 38.38 10)')))  ↩
    As geomA,
  ST_AsEWKT(ST_Shift_Longitude(ST_GeomFromEWKT('SRID=4326;POINT(241.42 38.38 10)'))) As  ↩
      geomb
geomA               geomB
----------              -----------
SRID=4326;POINT(241.42 38.38 10) SRID=4326;POINT(-118.58 38.38 10)

--regular line string
```

```
SELECT ST_AsText(ST_Shift_Longitude(ST_GeomFromText('LINESTRING(-118.58 38.38, -118.20  ↩
    38.45)')))

st_astext
----------
LINESTRING(241.42 38.38,241.8 38.45)
```

See Also

ST_GeomFromEWKT, ST_GeomFromText, ST_AsEWKT

7.9.16 ST_Simplify

ST_Simplify — Returns a "simplified" version of the given geometry using the Douglas-Peucker algorithm.

Synopsis

geometry **ST_Simplify**(geometry geomA, float tolerance);

Description

Returns a "simplified" version of the given geometry using the Douglas-Peucker algorithm. Will actually do something only with (multi)lines and (multi)polygons but you can safely call it with any kind of geometry. Since simplification occurs on a object-by-object basis you can also feed a GeometryCollection to this function.

Note
Note that returned geometry might loose its simplicity (see ST_IsSimple)

Note
Note topology may not be preserved and may result in invalid geometries. Use (see ST_SimplifyPreserveTopology) to preserve topology.

Performed by the GEOS module.

Availability: 1.2.2

Examples

A circle simplified too much becomes a triangle, medium an octagon,

```
SELECT ST_Npoints(the_geom) As np_before, ST_NPoints(ST_Simplify(the_geom,0.1)) As  ↩
    np01_notbadcircle, ST_NPoints(ST_Simplify(the_geom,0.5)) As np05_notquitecircle,
ST_NPoints(ST_Simplify(the_geom,1)) As np1_octagon, ST_NPoints(ST_Simplify(the_geom,10)) As ↩
     np10_triangle,
(ST_Simplify(the_geom,100) is null) As  np100_geometrygoesaway
FROM (SELECT ST_Buffer('POINT(1 3)', 10,12) As the_geom) As foo;
-result
 np_before | np01_notbadcircle | np05_notquitecircle | np1_octagon | np10_triangle |  ↩
     np100_geometrygoesaway
-----------+-------------------+---------------------+-------------+---------------+----------------

    49 |                33 |                    17 |            9 |             4 | t
```

See Also

ST_IsSimple, ST_SimplifyPreserveTopology

7.9.17 ST_SimplifyPreserveTopology

ST_SimplifyPreserveTopology — Returns a "simplified" version of the given geometry using the Douglas-Peucker algorithm. Will avoid creating derived geometries (polygons in particular) that are invalid.

Synopsis

geometry **ST_SimplifyPreserveTopology**(geometry geomA, float tolerance);

Description

Returns a "simplified" version of the given geometry using the Douglas-Peucker algorithm. Will avoid creating derived geometries (polygons in particular) that are invalid. Will actually do something only with (multi)lines and (multi)polygons but you can safely call it with any kind of geometry. Since simplification occurs on a object-by-object basis you can also feed a GeometryCollection to this function.

Performed by the GEOS module.

Note
Requires GEOS 3.0.0+

Availability: 1.3.3

Examples

Same example as Simplify, but we see Preserve Topology prevents oversimplification. The circle can at most become a square.

```
SELECT ST_Npoints(the_geom) As np_before, ST_NPoints(ST_SimplifyPreserveTopology(the_geom ↩
    ,0.1)) As np01_notbadcircle, ST_NPoints(ST_SimplifyPreserveTopology(the_geom,0.5)) As  ↩
    np05_notquitecircle,
ST_NPoints(ST_SimplifyPreserveTopology(the_geom,1)) As np1_octagon, ST_NPoints( ↩
    ST_SimplifyPreserveTopology(the_geom,10)) As np10_square,
ST_NPoints(ST_SimplifyPreserveTopology(the_geom,100)) As  np100_stillsquare
FROM (SELECT ST_Buffer('POINT(1 3)', 10,12) As the_geom) As foo;

--result--
 np_before | np01_notbadcircle | np05_notquitecircle | np1_octagon | np10_square |  ↩
     np100_stillsquare
-----------+-------------------+---------------------+-------------+---------------+----------------

    49 |                33 |                   17 |           9 |              5 |  ↩
                        5
```

See Also

ST_Simplify

7.9.18 ST_SymDifference

ST_SymDifference — Returns a geometry that represents the portions of A and B that do not intersect. It is called a symmetric difference because ST_SymDifference(A,B) = ST_SymDifference(B,A).

Synopsis

geometry **ST_SymDifference**(geometry geomA, geometry geomB);

Description

Returns a geometry that represents the portions of A and B that do not intersect. It is called a symmetric difference because ST_SymDifference(A,B) = ST_SymDifference(B,A). One can think of this as ST_Union(geomA,geomB) - ST_Intersection(A,B).

Performed by the GEOS module

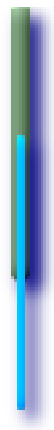
Note
Do not call with a GeometryCollection as an argument

✓ This method implements the OpenGIS 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 it seems to only consider x y when doing the difference and tacks back on the Z-Index

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

7.9.19 ST_Union

ST_Union — Returns a geometry that represents the point set union of the Geometries.

Synopsis

geometry **ST_Union**(geometry set g1field);
geometry **ST_Union**(geometry g1, geometry g2);
geometry **ST_Union**(geometry[] g1_array);

Description

Output type can be a MULTI* , single geometry, or Geometry Collection. Comes in 2 variants. Variant 1 unions 2 geometries resulting in a new geomety with no intersecting regions. Variant 2 is an aggregate function that takes a set of geometries and unions them into a single ST_Geometry resulting in no intersecting regions.

Aggregate version: This function returns a MULTI geometry or NON-MULTI geometry from a set 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.

Non-Aggregate version: This function returns a geometry being a union of two input geometries. Output type can be a MULTI* ,NON-MULTI or GEOMETRYCOLLECTION.

Note

ST_Collect and ST_Union are often interchangeable. ST_Union is in general orders of magnitude slower than ST_Collect because it tries to dissolve boundaries and reorder geometries to ensure that a constructed Multi* doesn't have intersecting regions.

Performed by the GEOS module.

NOTE: this function was formerly called GeomUnion(), which was renamed from "Union" because UNION is an SQL reserved word.

Availability: 1.4.0 - ST_Union was enhanced. ST_Union(geomarray) was introduced and also faster aggregate collection in PostgreSQL. If you are using GEOS 3.1.0+ ST_Union will use the faster Cascaded Union algorithm described in http://blog.cleverelephant.ca 2009/01/must-faster-unions-in-postgis-14.html

✓ This method implements the OpenGIS 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.

Examples

Aggregate example

```
SELECT stusps,
     ST_Multi(ST_Union(f.the_geom)) as singlegeom
   FROM sometable As f
GROUP BY stusps
```

Non-Aggregate example

```
SELECT ST_AsText(ST_Union(ST_GeomFromText('POINT(1 2)'),
  ST_GeomFromText('POINT(-2 3)') ) )

st_astext
----------
MULTIPOINT(-2 3,1 2)


SELECT ST_AsText(ST_Union(ST_GeomFromText('POINT(1 2)'),
    ST_GeomFromText('POINT(1 2)') ) );
st_astext
----------
POINT(1 2)

--3d example - sort of supports 3d (and with mixed dimensions!)
SELECT ST_AsEWKT(st_union(the_geom))
FROM
(SELECT ST_GeomFromEWKT('POLYGON((-7 4.2,-7.1 4.2,-7.1 4.3,
-7 4.2))') as the_geom
UNION ALL
SELECT ST_GeomFromEWKT('POINT(5 5 5)') as the_geom
UNION ALL
  SELECT ST_GeomFromEWKT('POINT(-2 3 1)') as the_geom
UNION ALL
SELECT ST_GeomFromEWKT('LINESTRING(5 5 5, 10 10 10)') as the_geom ) as foo;

st_asewkt
---------
```

```
GEOMETRYCOLLECTION(POINT(-2 3 1),LINESTRING(5 5 5,10 10 10),POLYGON((-7 4.2 5,-7.1 4.2  ↩
    5,-7.1 4.3 5,-7 4.2 5)));

--3d example not mixing dimensions
SELECT ST_AsEWKT(st_union(the_geom))
FROM
(SELECT ST_GeomFromEWKT('POLYGON((-7 4.2 2,-7.1 4.2 3,-7.1 4.3 2,
-7 4.2 2))') as the_geom
UNION ALL
SELECT ST_GeomFromEWKT('POINT(5 5 5)') as the_geom
UNION ALL
  SELECT ST_GeomFromEWKT('POINT(-2 3 1)') as the_geom
UNION ALL
SELECT ST_GeomFromEWKT('LINESTRING(5 5 5, 10 10 10)') as the_geom ) as foo;

st_asewkt
---------
GEOMETRYCOLLECTION(POINT(-2 3 1),LINESTRING(5 5 5,10 10 10),POLYGON((-7 4.2 2,-7.1 4.2  ↩
    3,-7.1 4.3 2,-7 4.2 2)))

--Examples using new Array construct
SELECT ST_Union(ARRAY(SELECT the_geom FROM sometable));

SELECT ST_AsText(ST_Union(ARRAY[ST_GeomFromText('LINESTRING(1 2, 3 4)'),
      ST_GeomFromText('LINESTRING(3 4, 4 5)')])) As wktunion;

--wktunion---
MULTILINESTRING((3 4,4 5),(1 2,3 4))
```

See Also

ST_Collect

7.10 Linear Referencing

7.10.1 ST_Line_Interpolate_Point

ST_Line_Interpolate_Point — Returns a point interpolated along a line. Second argument is a float8 between 0 and 1 representing fraction of total length of linestring the point has to be located.

Synopsis

geometry **ST_Line_Interpolate_Point**(geometry a_linestring, float a_fraction);

Description

Returns a point interpolated along a line. First argument must be a LINESTRING. Second argument is a float8 between 0 and 1 representing fraction of total linestring length the point has to be located.

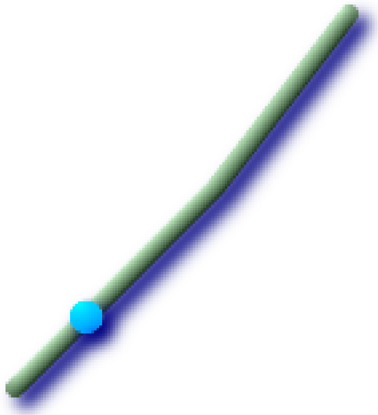
See ST_Line_Locate_Point for computing the line location nearest to a Point.

Note
Since release 1.1.1 this function also interpolates M and Z values (when present), while prior releases set them to 0.0.

Availability: 0.8.2, Z and M supported added in 1.1.1

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

Examples



A linestring with the interpolated point at 20% position (0.20)

```
--Return point 20% along 2d line
SELECT ST_AsEWKT(ST_Line_Interpolate_Point(the_line, 0.20))
  FROM (SELECT ST_GeomFromEWKT('LINESTRING(25 50, 100 125, 150 190)') as the_line) As foo;
   st_asewkt
----------------
 POINT(51.5974135047432 76.5974135047432)
```

```
--Return point mid-way of 3d line
SELECT ST_AsEWKT(ST_Line_Interpolate_Point(the_line, 0.5))
  FROM (SELECT ST_GeomFromEWKT('LINESTRING(1 2 3, 4 5 6, 6 7 8)') as the_line) As foo;

  st_asewkt
--------------------
 POINT(3.5 4.5 5.5)


--find closest point on a line to a point or other geometry
 SELECT ST_AsText(ST_Line_Interpolate_Point(foo.the_line, ST_Line_Locate_Point(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)
```

See Also

ST_AsText,ST_AsEWKT,ST_Length, ST_Line_Locate_Point

7.10.2 ST_Line_Locate_Point

ST_Line_Locate_Point — Returns a float between 0 and 1 representing the location of the closest point on LineString to the given Point, as a fraction of total 2d line length.

Synopsis

float **ST_Line_Locate_Point**(geometry a_linestring, geometry a_point);

Description

Returns a float between 0 and 1 representing the location of the closest point on LineString to the given Point, as a fraction of total 2d line length.

You can use the returned location to extract a Point (ST_Line_Interpolate_Point) or a substring (ST_Line_Substring).

This is useful for approximating numbers of addresses

Availability: 1.1.0

Examples

```
--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_Line_Locate_Point(street_line, house_loc) As integer) As street_num
FROM
(SELECT ST_GeomFromText('LINESTRING(1 2, 3 4)') As street_line,
  ST_MakePoint(x*1.01,y*1.03) As house_loc, 10 As startstreet_num,
    20 As endstreet_num
FROM generate_series(1,3) x CROSS JOIN generate_series(2,4) As y)
As foo
WHERE ST_DWithin(street_line, house_loc, 0.2);

 as_text_house_loc | street_num
-------------------+------------
 POINT(1.01 2.06)  |         10
 POINT(2.02 3.09)  |         15
 POINT(3.03 4.12)  |         20

 --find closest point on a line to a point or other geometry
 SELECT ST_AsText(ST_Line_Interpolate_Point(foo.the_line, ST_Line_Locate_Point(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)
```

See Also

ST_DWithin, ST_Length2D, ST_Line_Interpolate_Point, ST_Line_Substring

7.10.3 ST_Line_Substring

ST_Line_Substring — Return a linestring being a substring of the input one starting and ending at the given fractions of total 2d length. Second and third arguments are float8 values between 0 and 1.

Synopsis

geometry **ST_Line_Substring**(geometry a_linestring, float startfraction, float endfraction);

Description

Return a linestring being a substring of the input one starting and ending at the given fractions of total 2d length. Second and third arguments are float8 values between 0 and 1. This only works with LINESTRINGs. To use with contiguous MULTILINESTRINGs use in conjunction with ST_LineMerge.

If 'start' and 'end' have the same value this is equivalent to ST_Line_Interpolate_Point.

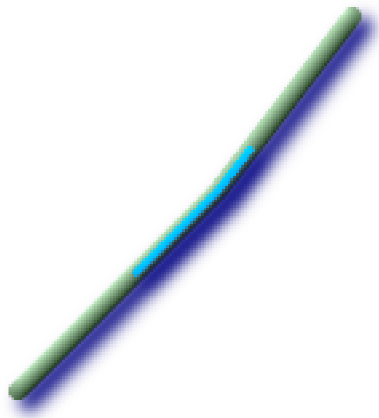
See ST_Line_Locate_Point for computing the line location nearest to a Point.

Note
Since release 1.1.1 this function also interpolates M and Z values (when present), while prior releases set them to unspecified values.

Availability: 1.1.0 , Z and M supported added in 1.1.1

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

Examples



A linestring seen with 1/3 midrange overlaid (0.333, 0.666)

```
--Return the approximate 1/3 mid-range part of a linestring
SELECT ST_AsText(ST_Line_SubString(ST_GeomFromText('LINESTRING(25 50, 100 125, 150 190)'),  ↩
    0.333, 0.666));
```

```
                       st_astext
------------------------------------------------------------------------------------------------ ↩

LINESTRING(69.2846934853974 94.2846934853974,100 125,111.700356260683 140.210463138888)

--The below example simulates a while loop in
--SQL using PostgreSQL generate_series() to cut all
--linestrings in a table to 100 unit segments
-- of which no segment is longer than 100 units
-- units are measured in the SRID units of measurement
-- It also assumes all geometries are LINESTRING or contiguous MULTILINESTRING
--and no geometry is longer than 100 units*10000
--for better performance you can reduce the 10000
--to match max number of segments you expect

SELECT field1, field2, ST_Line_Substring(the_geom, 100.00*n/length,
  CASE
  WHEN 100.00*(n+1) < length THEN 100.00*(n+1)/length
  ELSE 1
  END) As the_geom
FROM
  (SELECT sometable.field1, sometable.field2,
  ST_LineMerge(sometable.the_geom) AS the_geom,
  ST_Length(sometable.the_geom) As length
  FROM sometable
  ) AS t
CROSS JOIN generate_series(0,10000) AS n
WHERE n*100.00/length < 1;
```

See Also

ST_Length, ST_Line_Interpolate_Point, ST_LineMerge

7.10.4 ST_Locate_Along_Measure

ST_Locate_Along_Measure — Return a derived geometry collection value with elements that match the specified measure. Polygonal elements are not supported.

Synopsis

geometry **ST_Locate_Along_Measure**(geometry ageom_with_measure, float a_measure);

Description

Return a derived geometry collection value with elements that match the specified measure. Polygonal elements are not supported.

Semantic is specified by: ISO/IEC CD 13249-3:200x(E) - Text for Continuation CD Editing Meeting

Availability: 1.1.0

Note
Use this function only for geometries with an M component

This function supports M coordinates.

Examples

```
SELECT ST_AsEWKT(the_geom)
    FROM
    (SELECT ST_Locate_Along_Measure(
      ST_GeomFromEWKT(’MULTILINESTRINGM((1 2 3, 3 4 2, 9 4 3),
    (1 2 3, 5 4 5))’),3) As the_geom) As foo;

             st_asewkt
-----------------------------------------------------------
 GEOMETRYCOLLECTIONM(MULTIPOINT(1 2 3,9 4 3),POINT(1 2 3))

--Geometry collections are difficult animals so dump them
--to make them more digestable
SELECT ST_AsEWKT((ST_Dump(the_geom)).geom)
  FROM
  (SELECT ST_Locate_Along_Measure(
      ST_GeomFromEWKT(’MULTILINESTRINGM((1 2 3, 3 4 2, 9 4 3),
  (1 2 3, 5 4 5))’),3) As the_geom) As foo;

   st_asewkt
---------------
 POINTM(1 2 3)
 POINTM(9 4 3)
 POINTM(1 2 3)
```

See Also

ST_Dump, ST_Locate_Between_Measures

7.10.5 ST_Locate_Between_Measures

ST_Locate_Between_Measures — Return a derived geometry collection value with elements that match the specified range of measures inclusively. Polygonal elements are not supported.

Synopsis

geometry **ST_Locate_Between_Measures**(geometry geomA, float measure_start, float measure_end);

Description

Return a derived geometry collection value with elements that match the specified range of measures inclusively. Polygonal elements are not supported.

Semantic is specified by: ISO/IEC CD 13249-3:200x(E) - Text for Continuation CD Editing Meeting

Availability: 1.1.0

✓ This function supports M coordinates.

Examples

```
SELECT ST_AsEWKT(the_geom)
    FROM
    (SELECT ST_Locate_Between_Measures(
      ST_GeomFromEWKT(’MULTILINESTRINGM((1 2 3, 3 4 2, 9 4 3),
    (1 2 3, 5 4 5))’),1.5, 3) As the_geom) As foo;

               st_asewkt
-----------------------------------------------------------------
 GEOMETRYCOLLECTIONM(LINESTRING(1 2 3,3 4 2,9 4 3),POINT(1 2 3))

--Geometry collections are difficult animals so dump them
--to make them more digestable
SELECT ST_AsEWKT((ST_Dump(the_geom)).geom)
    FROM
    (SELECT ST_Locate_Between_Measures(
      ST_GeomFromEWKT(’MULTILINESTRINGM((1 2 3, 3 4 2, 9 4 3),
    (1 2 3, 5 4 5))’),1.5, 3) As the_geom) As foo;

       st_asewkt
--------------------------------
 LINESTRINGM(1 2 3,3 4 2,9 4 3)
 POINTM(1 2 3)
```

See Also

ST_Dump, ST_Locate_Along_Measure

7.10.6 ST_LocateBetweenElevations

ST_LocateBetweenElevations — Return a derived geometry (collection) value with elements that intersect the specified range of elevations inclusively. Only 3D, 4D LINESTRINGS and MULTILINESTRINGS are supported.

Synopsis

geometry **ST_LocateBetweenElevations**(geometry geom_mline, float elevation_start, float elevation_end);

Description

Return a derived geometry (collection) value with elements that intersect the specified range of elevations inclusively. Only 3D, 3DM LINESTRINGS and MULTILINESTRINGS are supported.

Availability: 1.4.0

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

Examples

```
SELECT ST_AsEWKT(ST_LocateBetweenElevations(
      ST_GeomFromEWKT(’LINESTRING(1 2 3, 4 5 6)’),2,4)) As ewelev;
                   ewelev
----------------------------------------------------------------
  MULTILINESTRING((1 2 3,2 3 4))

SELECT ST_AsEWKT(ST_LocateBetweenElevations(
      ST_GeomFromEWKT(’LINESTRING(1 2 6, 4 5 -1, 7 8 9)’),6,9)) As ewelev;

        ewelev
----------------------------------------------------------------
GEOMETRYCOLLECTION(POINT(1 2 6),LINESTRING(6.1 7.1 6,7 8 9))

--Geometry collections are difficult animals so dump them
--to make them more digestable
SELECT ST_AsEWKT((ST_Dump(the_geom)).geom)
    FROM
    (SELECT ST_LocateBetweenElevations(
      ST_GeomFromEWKT(’LINESTRING(1 2 6, 4 5 -1, 7 8 9)’),6,9) As the_geom) As foo;

       st_asewkt
--------------------------------
POINT(1 2 6)
LINESTRING(6.1 7.1 6,7 8 9)
```

See Also

ST_Dump

7.10.7 ST_AddMeasure

ST_AddMeasure — Return a derived geometry with measure elements linearly interpolated between the start and end points. If the geometry has no measure dimension, one is added. If the geometry has a measure dimension, it is over-written with new values. Only LINESTRINGS and MULTILINESTRINGS are supported.

Synopsis

geometry **ST_AddMeasure**(geometry geom_mline, float measure_start, float measure_end);

Description

Return a derived geometry with measure elements linearly interpolated between the start and end points. If the geometry has no measure dimension, one is added. If the geometry has a measure dimension, it is over-written with new values. Only LINESTRINGS and MULTILINESTRINGS are supported.

Availability: 1.5.0

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

Examples

```
SELECT ST_AsEWKT(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_AsEWKT(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_AsEWKT(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_AsEWKT(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))
```

7.11 Long Transactions Support

This module and associated pl/pgsql functions have been implemented to provide long locking support required by Web Feature Service specification.

Note
Users must use serializable transaction level otherwise locking mechanism would break.

7.11.1 AddAuth

AddAuth — Add an authorization token to be used in current transaction.

Synopsis

boolean **AddAuth**(text auth_token);

Description

Add an authorization token to be used in current transaction.

Creates/adds to a temp table called temp_lock_have_table the current transaction identifier and authorization token key.

Availability: 1.1.3

Examples

```
    SELECT LockRow('towns', '353', 'priscilla');
    BEGIN TRANSACTION;
      SELECT AddAuth('joey');
      UPDATE towns SET the_geom = ST_Translate(the_geom,2,2) WHERE gid = 353;
    COMMIT;


    ---Error--
    ERROR:  UPDATE where "gid" = '353' requires authorization 'priscilla'
```

See Also

LockRow

7.11.2 CheckAuth

CheckAuth — Creates trigger on a table to prevent/allow updates and deletes of rows based on authorization token.

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);

Description

Creates trigger on a table to prevent/allow updates and deletes of rows based on authorization token. Identify rows using <rowid_col> column.

If a_schema_name is not passed in, then searches for table in current schema.

Note
If an authorization trigger already exists on this table function errors.
If Transaction support is not enabled, function throws an exception.

Availability: 1.1.3

Examples

```
SELECT CheckAuth('public', 'towns', 'gid');
result
------
0
```

See Also

EnableLongTransactions

7.11.3 DisableLongTransactions

DisableLongTransactions — Disable long transaction support. This function removes the long transaction support metadata tables, and drops all triggers attached to lock-checked tables.

Synopsis

text **DisableLongTransactions**

Description

Disable long transaction support. This function removes the long transaction support metadata tables, and drops all triggers attached to lock-checked tables.

Drops meta table called `authorization_table` and a view called `authorized_tables` and all triggers called `checkauthtrigger`

Availability: 1.1.3

Examples

```
SELECT DisableLongTransactions();
--result--
Long transactions support disabled
```

See Also

EnableLongTransactions

7.11.4 EnableLongTransactions

EnableLongTransactions — Enable long transaction support. This function creates the required metadata tables, needs to be called once before using the other functions in this section. Calling it twice is harmless.

Synopsis

text **EnableLongTransactions**

Description

Enable long transaction support. This function creates the required metadata tables, needs to be called once before using the other functions in this section. Calling it twice is harmless.

Creates a meta table called `authorization_table` and a view called `authorized_tables`

Availability: 1.1.3

Examples

```
SELECT EnableLongTransactions();
--result--
Long transactions support enabled
```

See Also

DisableLongTransactions

7.11.5 LockRow

LockRow — Set lock/authorization for specific row in 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);

Description

Set lock/authorization for specific row in table <authid> is a text value, <expires> is a timestamp defaulting to now()+1hour. Returns 1 if lock has been assigned, 0 otherwise (already locked by other auth)

Availability: 1.1.3

Examples

```
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
```

See Also

UnlockRows

7.11.6 UnlockRows

UnlockRows — Remove all locks held by specified authorization id. Returns the number of locks released.

Synopsis

integer **UnlockRows**(text auth_token);

Description

Remove all locks held by specified authorization id. Returns the number of locks released.

Availability: 1.1.3

Examples

```
    SELECT LockRow('towns', '353', 'priscilla');
    SELECT LockRow('towns', '2', 'priscilla');
    SELECT UnLockRows('priscilla');
    UnLockRows
    ------------
    2
```

See Also

LockRow

7.12 Miscellaneous Functions

7.12.1 ST_Accum

ST_Accum — Aggregate. Constructs an array of geometries.

Synopsis

geometry[] **ST_Accum**(geometry set geomfield);

Description

Aggregate. Constructs an array of geometries.

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

✓ This method supports Circular Strings and Curves

Examples

```
SELECT (ST_Accum(the_geom)) As all_em, ST_AsText((ST_Accum(the_geom))[1]) As grabone,
(ST_Accum(the_geom))[2:4] as grab_rest
      FROM (SELECT ST_MakePoint(a*CAST(random()*10 As integer), a*CAST(random()*10 As  ↩
          integer), a*CAST(random()*10 As integer)) As the_geom
        FROM generate_series(1,4) a) As foo;

all_em|grabone   | grab_rest

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

 {0101000080000000000000144000000000000024400000000000001040:
 0101000080000000000
0001840000000000000002C400000000000003040:
0101000080000000000000354000000000000038400000000000001840:
0101000080000000000000404000000000000003C400000000000003040} |
 POINT(5 10) | {01010000800000000000000184000000000000002C400000000000003040:
 0101000080000000000000354000000000000038400000000000001840:
 0101000080000000000000404000000000000003C400000000000003040}
(1 row)
```

See Also

ST_Collect

7.12.2 Box2D

Box2D — Returns a BOX2D representing the maximum extents of the geometry.

Synopsis

box2d **Box2D**(geometry geomA);

Description

Returns a BOX2D representing the maximum extents of the geometry.

✓ This method supports Circular Strings and Curves

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

7.12.3 Box3D

Box3D — Returns a BOX3D representing the maximum extents of the geometry.

Synopsis

box3d **Box3D**(geometry geomA);

Description

Returns a BOX3D representing the maximum extents of the geometry.

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

✓ This method supports Circular Strings and Curves

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

7.12.4 ST_Estimated_Extent

ST_Estimated_Extent — Return the 'estimated' extent of the given spatial table. The estimated is taken from the geometry column's statistics. The current schema will be used if not specified.

Synopsis

box2d **ST_Estimated_Extent**(text schema_name, text table_name, text geocolumn_name);
box2d **ST_Estimated_Extent**(text table_name, text geocolumn_name);

Description

Return the 'estimated' extent of the given spatial table. The estimated is taken from the geometry column's statistics. The current schema will be used if not specified.

For PostgreSQL>=8.0.0 statistics are gathered by VACUUM ANALYZE and resulting extent will be about 95% of the real one.

Note
In absence of statistics (empty table or no ANALYZE called) this function returns NULL. Prior to version 1.5.4 an exception was thrown instead.

For PostgreSQL<8.0.0 statistics are gathered by update_geometry_stats() and resulting extent will be exact.

Availability: 1.0.0

✓ This method supports Circular Strings and Curves

Examples

```
SELECT ST_Estimated_extent('ny', 'edges', 'the_geom');
--result--
BOX(-8877653 4912316,-8010225.5 5589284)

SELECT ST_Estimated_Extent('feature_poly', 'the_geom');
--result--
BOX(-124.659652709961 24.6830825805664,-67.7798080444336 49.0012092590332)
```

See Also

ST_Extent

7.12.5 ST_Expand

ST_Expand — Returns bounding box expanded in all directions from the bounding box of the input geometry. Uses double-precision

Synopsis

geometry **ST_Expand**(geometry g1, float units_to_expand);
box2d **ST_Expand**(box2d g1, float units_to_expand);
box3d **ST_Expand**(box3d g1, float units_to_expand);

Description

This function returns a bounding box expanded in all directions from the bounding box of the input geometry, by an amount specified in the second argument. Uses double-precision. Very useful for distance() queries, or bounding box queries to add an index filter to the query.

There are 3 variants of this. The one that takes a geometry will return a POLYGON geometry representation of the bounding box and is the most commonly used variant.

ST_Expand is similar in concept to ST_Buffer except while buffer expands the geometry in all directions, ST_Expand expands the bounding box an x,y,z unit amount.

Units are in the units of the spatial reference system in use denoted by the SRID

Note
Pre 1.3, ST_Expand was used in conjunction with distance to do indexable queries. Something of the form `the_geom && ST_Expand(’POINT(10 20)’, 10) AND ST_Distance(the_geom, ’POINT(10 20)’) < 10` Post 1.2, this was replaced with the easier ST_DWithin construct.

Note
Bounding boxes of all geometries are currently 2-d even if they are 3-dimensional geometries.

Note
Availability: 1.5.0 behavior changed to output double precision instead of float4 coordinates.

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_AsEWKT, ST_Buffer, ST_DWithin, ST_GeomFromEWKT,ST_GeomFromText, ST_SRID

7.12.6 ST_Extent

ST_Extent — an aggregate function that returns the bounding box that bounds rows of geometries.

Synopsis

box3d_extent **ST_Extent**(geometry set geomfield);

Description

ST_Extent returns a bounding box that encloses a set of geometries. The ST_Extent function is an "aggregate" function in the terminology of SQL. That means that it operates on lists of data, in the same way the SUM() and AVG() functions do.

Since it returns a bounding box, the spatial Units are in the units of the spatial reference system in use denoted by the SRID

ST_Extent is similar in concept to Oracle Spatial/Locator's SDO_AGGR_MBR

Note
Since ST_Extent returns a bounding box, the SRID meta-data is lost. Use ST_SetSRID to force it back into a geometry with SRID meta data. The coordinates are in the units of the spatial ref of the orginal geometries.

Note
ST_Extent will return boxes with only an x and y component even with (x,y,z) coordinate geometries. To maintain x,y,z use ST_Extent3D instead.

Note
Availability: 1.4.0 As of 1.4.0 now returns a box3d_extent instead of box2d object.

Examples

Note
Examples below use Massachusetts State Plane ft (SRID=2249)

```
SELECT ST_Extent(the_geom) as bextent FROM sometable;
           st_bextent
------------------------------------
BOX(739651.875 2908247.25,794875.8125 2970042.75)


--Return extent of each category of geometries
SELECT ST_Extent(the_geom) as bextent
FROM sometable
GROUP BY category ORDER BY category;

            bextent                       |         name
----------------------------------------------------+----------------
 BOX(778783.5625 2951741.25,794875.8125 2970042.75) | A
 BOX(751315.8125 2919164.75,765202.6875 2935417.25) | B
 BOX(739651.875 2917394.75,756688.375 2935866)      | C

 --Force back into a geometry
 -- and render the extended text representation of that geometry
SELECT ST_SetSRID(ST_Extent(the_geom),2249) as bextent FROM sometable;

        bextent
--------------------------------------------------------------------------------
 SRID=2249;POLYGON((739651.875 2908247.25,739651.875 2970042.75,794875.8125 2970042.75,
 794875.8125 2908247.25,739651.875 2908247.25))
```

See Also

ST_AsEWKT, ST_Extent3D, ST_SetSRID, ST_SRID

7.12.7 ST_Extent3D

ST_Extent3D — an aggregate function that returns the box3D bounding box that bounds rows of geometries.

Synopsis

box3d **ST_Extent3D**(geometry set geomfield);

Description

ST_Extent3D returns a box3d (includes Z coordinate) bounding box that encloses a set of geometries. The ST_Extent3D function is an "aggregate" function in the terminology of SQL. That means that it operates on lists of data, in the same way the SUM() and AVG() functions do.

Since it returns a bounding box, the spatial Units are in the units of the spatial reference system in use denoted by the SRID

Note
Since ST_Extent3D returns a bounding box, the SRID meta-data is lost. Use ST_SetSRID to force it back into a geometry with SRID meta data. The coordinates are in the units of the spatial ref of the orginal geometries.

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

✓ This method supports Circular Strings and Curves

Examples

```
SELECT ST_Extent3D(foo.the_geom) As b3extent
FROM (SELECT ST_MakePoint(x,y,z) As the_geom
  FROM generate_series(1,3) As x
    CROSS JOIN generate_series(1,2) As y
    CROSS JOIN generate_series(0,2) As Z) As foo;
    b3extent
--------------------
 BOX3D(1 1 0,3 2 2)

--Get the extent of various elevated circular strings
SELECT ST_Extent3D(foo.the_geom) As b3extent
FROM (SELECT ST_Translate(ST_Force_3DZ(ST_LineToCurve(ST_Buffer(ST_MakePoint(x,y),1))),0,0, ↩
    z) As the_geom
  FROM generate_series(1,3) As x
    CROSS JOIN generate_series(1,2) As y
    CROSS JOIN generate_series(0,2) As Z) As foo;

  b3extent
--------------------
 BOX3D(1 0 0,4 2 2)
```

See Also

ST_Extent, ST_Force_3DZ

7.12.8 Find_SRID

Find_SRID — The syntax is find_srid(<db/schema>, <table>, <column>) and the function returns the integer SRID of the specified column by searching through the GEOMETRY_COLUMNS table.

Synopsis

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

Description

The syntax is find_srid(<db/schema>, <table>, <column>) and the function returns the integer SRID of the specified column by searching through the GEOMETRY_COLUMNS table. If the geometry column has not been properly added with the AddGeometryColumns() function, this function will not work either.

Examples

```
 SELECT Find_SRID('public', 'tiger_us_state_2007', 'the_geom_4269');
find_srid
----------
4269
```

See Also

ST_SRID

7.12.9 ST_Mem_Size

ST_Mem_Size — Returns the amount of space (in bytes) the geometry takes.

Synopsis

integer **ST_Mem_Size**(geometry geomA);

Description

Returns the amount of space (in bytes) the geometry takes.

This is a nice compliment to PostgreSQL built in functions pg_size_pretty, pg_relation_size, pg_total_relation_size.

> **Note**
> pg_relation_size which gives the byte size of a table may return byte size lower than ST_Mem_Size. 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.

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

✓ This method supports Circular Strings and Curves

Examples

```
--Return how much byte space Boston takes up  in our Mass data set
SELECT pg_size_pretty(SUM(ST_Mem_Size(the_geom))) as totgeomsum,
pg_size_pretty(SUM(CASE WHEN town = 'BOSTON' THEN st_mem_size(the_geom) ELSE 0 END)) As  ↩
    bossum,
CAST(SUM(CASE WHEN town = 'BOSTON' THEN st_mem_size(the_geom) ELSE 0 END)*1.00 /
    SUM(st_mem_size(the_geom))*100 As numeric(10,2)) As perbos
FROM towns;

totgeomsum  bossum  perbos
----------  ------  ------
1522 kB   30 kB 1.99


SELECT ST_Mem_Size(ST_GeomFromText('CIRCULARSTRING(220268 150415,220227 150505,220227  ↩
    150406)'));
```

```
---
73

--What percentage of our table is taken up by just the geometry
SELECT pg_total_relation_size('public.neighborhoods') As fulltable_size, sum(ST_Mem_Size( ↩
    the_geom)) As geomsize,
sum(ST_Mem_Size(the_geom))*1.00/pg_total_relation_size('public.neighborhoods')*100 As  ↩
    pergeom
FROM neighborhoods;
fulltable_size geomsize  pergeom
------------------------------------------------
262144         96238   36.71188354492187500000
```

See Also

7.12.10 ST_Point_Inside_Circle

ST_Point_Inside_Circle — Is the point geometry insert circle defined by center_x, center_y , radius

Synopsis

boolean **ST_Point_Inside_Circle**(geometry a_point, float center_x, float center_y, float radius);

Description

The syntax for this functions is point_inside_circle(<geometry>,<circle_center_x>,<circle_center_y>,<radius>). Returns the true if the geometry is a point and is inside the circle. Returns false otherwise.

Note This only works for points as the name suggests

Examples

```
SELECT ST_Point_Inside_Circle(ST_Point(1,2), 0.5, 2, 3);
 st_point_inside_circle
------------------------
 t
```

See Also

ST_DWithin

7.12.11 ST_XMax

ST_XMax — Returns X maxima of a bounding box 2d or 3d or a geometry.

Synopsis

float **ST_XMax**(box3d aGeomorBox2DorBox3D);

Description

Returns X maxima of a bounding box 2d or 3d or a geometry.

Note
Although this function is only defined for box3d, it will work for box2d and geometry because of the auto-casting behavior defined for geometries and box2d. However you can not feed it a geometry or box2d text represenation, since that will 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 autocast the string representation to a ↩
     BOX3D
SELECT ST_XMax('LINESTRING(1 3, 5 6)');

--ERROR:  BOX3D parser - doesnt start with BOX3D(

SELECT ST_XMax(ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227  ↩
    150406 3)'));
st_xmax
--------
220288.248780547
```

See Also

ST_XMin, ST_YMax, ST_YMin, ST_ZMax, ST_ZMin

7.12.12 ST_XMin

ST_XMin — Returns X minima of a bounding box 2d or 3d or a geometry.

Synopsis

float **ST_XMin**(box3d aGeomorBox2DorBox3D);

Description

Returns X minima of a bounding box 2d or 3d or a geometry.

Note
Although this function is only defined for box3d, it will work for box2d and geometry because of the auto-casting behavior defined for geometries and box2d. However you can not feed it a geometry or box2d text represenation, since that will 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 autocast the string representation to a ↩
     BOX3D
SELECT ST_XMin('LINESTRING(1 3, 5 6)');

--ERROR:  BOX3D parser - doesnt start with BOX3D(

SELECT ST_XMin(ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227  ↩
    150406 3)'));
st_xmin
--------
220186.995121892
```

See Also

ST_XMax, ST_YMax, ST_YMin, ST_ZMax, ST_ZMin

7.12.13 ST_YMax

ST_YMax — Returns Y maxima of a bounding box 2d or 3d or a geometry.

Synopsis

float **ST_YMax**(box3d aGeomorBox2DorBox3D);

Description

Returns Y maxima of a bounding box 2d or 3d or a geometry.

Note
Although this function is only defined for box3d, it will work for box2d and geometry because of the auto-casting behavior defined for geometries and box2d. However you can not feed it a geometry or box2d text represenation, since that will 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 autocast the string representation to a ↩
     BOX3D
SELECT ST_YMax('LINESTRING(1 3, 5 6)');

--ERROR:  BOX3D parser - doesnt start with BOX3D(

SELECT ST_YMax(ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227  ↩
    150406 3)'));
st_ymax
--------
150506.126829327
```

See Also

ST_XMin, ST_XMax, ST_YMin, ST_ZMax, ST_ZMin

7.12.14 ST_YMin

ST_YMin — Returns Y minima of a bounding box 2d or 3d or a geometry.

Synopsis

float **ST_YMin**(box3d aGeomorBox2DorBox3D);

Description

Returns Y minima of a bounding box 2d or 3d or a geometry.

Note
Although this function is only defined for box3d, it will work for box2d and geometry because of the auto-casting behavior defined for geometries and box2d. However you can not feed it a geometry or box2d text represenation, since that will 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 autocast the string representation to a ↩
     BOX3D
SELECT ST_YMin('LINESTRING(1 3, 5 6)');

--ERROR:  BOX3D parser - doesnt start with BOX3D(

SELECT ST_YMin(ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227  ↩
    150406 3)'));
st_ymin
--------
150406
```

See Also

ST_GeomFromEWKT, ST_XMin, ST_XMax, ST_YMax, ST_ZMax, ST_ZMin

7.12.15 ST_ZMax

ST_ZMax — Returns Z minima of a bounding box 2d or 3d or a geometry.

Synopsis

float **ST_ZMax**(box3d aGeomorBox2DorBox3D);

Description

Returns Z maxima of a bounding box 2d or 3d or a geometry.

Note
Although this function is only defined for box3d, it will work for box2d and geometry because of the auto-casting behavior defined for geometries and box2d. However you can not feed it a geometry or box2d text represenation, since that will 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 autocast the string representation to a ↩
     BOX3D
SELECT ST_ZMax('LINESTRING(1 3 4, 5 6 7)');

--ERROR:  BOX3D parser - doesnt start with BOX3D(

SELECT ST_ZMax(ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227  ↩
    150406 3)'));
st_zmax
--------
3
```

See Also

ST_GeomFromEWKT, ST_XMin, ST_XMax, ST_YMax, ST_YMin, ST_ZMax

7.12.16 ST_ZMin

ST_ZMin — Returns Z minima of a bounding box 2d or 3d or a geometry.

Synopsis

float **ST_ZMin**(box3d aGeomorBox2DorBox3D);

Description

Returns Z minima of a bounding box 2d or 3d or a geometry.

Note
Although this function is only defined for box3d, it will work for box2d and geometry because of the auto-casting behavior defined for geometries and box2d. However you can not feed it a geometry or box2d text represenation, since that will 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 autocast the string representation to a ↩
     BOX3D
SELECT ST_ZMin('LINESTRING(1 3 4, 5 6 7)');

--ERROR:  BOX3D parser - doesnt start with BOX3D(

SELECT ST_ZMin(ST_GeomFromEWKT('CIRCULARSTRING(220268 150415 1,220227 150505 2,220227  ↩
    150406 3)'));
st_zmin
--------
1
```

See Also

ST_GeomFromEWKT, ST_GeomFromText, ST_XMin, ST_XMax, ST_YMax, ST_YMin, ST_ZMax

7.13 Exceptional Functions

These functions are rarely used functions that should only be used if your data is corrupted in someway. They are used for troubleshooting corruption and also fixing things that should under normal circumstances, never happen.

7.13.1 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 the_geom =  PostGIS_AddBBox(the_geom)
 WHERE PostGIS_HasBBox(the_geom) = false;
```

See Also

PostGIS_DropBBox, PostGIS_HasBBox

7.13.2 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 the_geom =  PostGIS_DropBBox(the_geom)
 WHERE Not (Box2D(ST_AsBinary(the_geom)) = Box2D(the_geom));

  UPDATE sometable
 SET the_geom =  PostGIS_AddBBox(the_geom)
 WHERE Not PostGIS_HasBBOX(the_geom);
```

See Also

PostGIS_AddBBox, PostGIS_HasBBox, Box2D

7.13.3 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 the_geom
FROM sometable WHERE PostGIS_HasBBox(the_geom) = false;
```

See Also

PostGIS_AddBBox, PostGIS_DropBBox

Chapter 8

PostGIS Special Functions Index

8.1 PostGIS Aggregate Functions

The functions given below are spatial aggregate functions provided with PostGIS that can be used just like any other sql aggregate function such as sum, average.

- ST_Accum - Aggregate. Constructs an array of geometries.
- ST_Collect - Return a specified ST_Geometry value from a collection of other geometries.
- ST_Extent - an aggregate function that returns the bounding box that bounds rows of geometries.
- ST_Extent3D - an aggregate function that returns the box3D bounding box that bounds rows of geometries.
- ST_MakeLine - Creates a Linestring from point geometries.
- ST_MemUnion - Same as ST_Union, only memory-friendly (uses less memory and more processor time).
- ST_Polygonize - Aggregate. Creates a GeometryCollection containing possible polygons formed from the constituent linework of a set of geometries.
- ST_Union - Returns a geometry that represents the point set union of the Geometries.

8.2 PostGIS SQL-MM Compliant Functions

The functions given below are PostGIS functions that conform to the SQL/MM 3 standard

Note
SQL-MM defines the default SRID of all geometry constructors as 0. PostGIS uses a default SRID of -1.

- ST_Area - Returns the area of the surface if it is a polygon or multi-polygon. For "geometry" type area is in SRID units. For "geography" area is in square meters. This method implements the SQL/MM specification. SQL-MM 3: 8.1.2, 9.5.3
- ST_AsBinary - Return the Well-Known Binary (WKB) representation of the geometry/geography without SRID meta data. This method implements the SQL/MM specification. SQL-MM 3: 5.1.37
- ST_AsText - Return the Well-Known Text (WKT) representation of the geometry/geography without SRID metadata. This method implements the SQL/MM specification. SQL-MM 3: 5.1.25

- ST_Boundary - Returns the closure of the combinatorial boundary of this Geometry. This method implements the SQL/MM specification. SQL-MM 3: 5.1.14
- ST_Buffer - (T) For geometry: Returns a geometry that represents all points whose distance from this Geometry is less than or equal to distance. Calculations are in the Spatial Reference System of this Geometry. For geography: Uses a planar transform wrapper. Introduced in 1.5 support for different end cap and mitre settings to control shape. buffer_style options: quad_segs=#,endcap=round|flat|square,join=round|mitre|bevel,mitre_limit=#.# This method implements the SQL/MM specification. SQL-MM 3: 5.1.17
- ST_Centroid - Returns the geometric center of a geometry. This method implements the SQL/MM specification. SQL-MM 3: 8.1.4, 9.5.5
- ST_Contains - Returns true if and only if no points of B lie in the exterior of A, and at least one point of the interior of B lies in the interior of A. This method implements the SQL/MM specification. SQL-MM 3: 5.1.31
- ST_ConvexHull - The convex hull of a geometry represents the minimum convex geometry that encloses all geometries within the set. This method implements the SQL/MM specification. SQL-MM 3: 5.1.16
- ST_CoordDim - Return the coordinate dimension of the ST_Geometry value. This method implements the SQL/MM specification. SQL-MM 3: 5.1.3
- ST_Crosses - Returns TRUE if the supplied geometries have some, but not all, interior points in common. This method implements the SQL/MM specification. SQL-MM 3: 5.1.29
- ST_CurveToLine - Converts a CIRCULARSTRING/CURVEDPOLYGON to a LINESTRING/POLYGON This method implements the SQL/MM specification. SQL-MM 3: 7.1.7
- ST_Difference - Returns a geometry that represents that part of geometry A that does not intersect with geometry B. This method implements the SQL/MM specification. SQL-MM 3: 5.1.20
- ST_Dimension - The inherent dimension of this Geometry object, which must be less than or equal to the coordinate dimension. This method implements the SQL/MM specification. SQL-MM 3: 5.1.2
- ST_Disjoint - Returns TRUE if the Geometries do not "spatially intersect" - if they do not share any space together. This method implements the SQL/MM specification. SQL-MM 3: 5.1.26
- ST_Distance - For geometry type Returns the 2-dimensional cartesian minimum distance (based on spatial ref) between two geometries in projected units. For geography type defaults to return spheroidal minimum distance between two geographies in meters. This method implements the SQL/MM specification. SQL-MM 3: 5.1.23
- ST_EndPoint - Returns the last point of a LINESTRING geometry as a POINT. This method implements the SQL/MM specification. SQL-MM 3: 7.1.4
- ST_Envelope - Returns a geometry representing the double precision (float8) bounding box of the supplied geometry. This method implements the SQL/MM specification. SQL-MM 3: 5.1.15
- ST_Equals - Returns true if the given geometries represent the same geometry. Directionality is ignored. This method implements the SQL/MM specification. SQL-MM 3: 5.1.24
- ST_ExteriorRing - Returns a line string representing the exterior ring of the POLYGON geometry. Return NULL if the geometry is not a polygon. Will not work with MULTIPOLYGON This method implements the SQL/MM specification. SQL-MM 3: 8.2.3, 8.3.3
- ST_GMLToSQL - Return a specified ST_Geometry value from GML representation. This is an alias name for ST_GeomFromGML This method implements the SQL/MM specification. SQL-MM 3: 5.1.50 (except for curves support).
- ST_GeomCollFromText - Makes a collection Geometry from collection WKT with the given SRID. If SRID is not give, it defaults to -1. This method implements the SQL/MM specification.
- ST_GeomFromText - Return a specified ST_Geometry value from Well-Known Text representation (WKT). This method implements the SQL/MM specification. SQL-MM 3: 5.1.40
- ST_GeomFromWKB - Creates a geometry instance from a Well-Known Binary geometry representation (WKB) and optional SRID. This method implements the SQL/MM specification. SQL-MM 3: 5.1.41

- ST_GeometryFromText - Return a specified ST_Geometry value from Well-Known Text representation (WKT). This is an alias name for ST_GeomFromText This method implements the SQL/MM specification. SQL-MM 3: 5.1.40
- ST_GeometryN - Return the 1-based Nth geometry if the geometry is a GEOMETRYCOLLECTION, MULTIPOINT, MULTILINESTRING, MULTICURVE or MULTIPOLYGON. Otherwise, return NULL. This method implements the SQL/MM specification. SQL-MM 3: 9.1.5
- ST_GeometryType - Return the geometry type of the ST_Geometry value. This method implements the SQL/MM specification. SQL-MM 3: 5.1.4
- ST_InteriorRingN - Return the Nth interior linestring ring of the polygon geometry. Return NULL if the geometry is not a polygon or the given N is out of range. This method implements the SQL/MM specification. SQL-MM 3: 8.2.6, 8.3.5
- ST_Intersection - (T) Returns a geometry that represents the shared portion of geomA and geomB. The geography implementation does a transform to geometry to do the intersection and then transform back to WGS84. This method implements the SQL/MM specification. SQL-MM 3: 5.1.18
- ST_Intersects - Returns TRUE if the Geometries/Geography "spatially intersect" - (share any portion of space) and FALSE if they don't (they are Disjoint). For geography -- tolerance is 0.00001 meters (so any points that close are considered to intersect) This method implements the SQL/MM specification. SQL-MM 3: 5.1.27
- ST_IsClosed - Returns TRUE if the LINESTRING's start and end points are coincident. This method implements the SQL/MM specification. SQL-MM 3: 7.1.5, 9.3.3
- ST_IsEmpty - Returns true if this Geometry is an empty geometry . If true, then this Geometry represents the empty point set - i.e. GEOMETRYCOLLECTION(EMPTY). This method implements the SQL/MM specification. SQL-MM 3: 5.1.7
- ST_IsRing - Returns TRUE if this LINESTRING is both closed and simple. This method implements the SQL/MM specification. SQL-MM 3: 7.1.6
- ST_IsSimple - Returns (TRUE) if this Geometry has no anomalous geometric points, such as self intersection or self tangency. This method implements the SQL/MM specification. SQL-MM 3: 5.1.8
- ST_IsValid - Returns true if the ST_Geometry is well formed. This method implements the SQL/MM specification. SQL-MM 3: 5.1.9
- ST_Length - Returns the 2d length of the geometry if it is a linestring or multilinestring. geometry are in units of spatial reference and geography are in meters (default spheroid) This method implements the SQL/MM specification. SQL-MM 3: 7.1.2, 9.3.4
- ST_LineFromText - Makes a Geometry from WKT representation with the given SRID. If SRID is not given, it defaults to -1. This method implements the SQL/MM specification. SQL-MM 3: 7.2.8
- ST_LineFromWKB - Makes a LINESTRING from WKB with the given SRID This method implements the SQL/MM specification. SQL-MM 3: 7.2.9
- ST_LinestringFromWKB - Makes a geometry from WKB with the given SRID. This method implements the SQL/MM specification. SQL-MM 3: 7.2.9
- ST_M - Return the M coordinate of the point, or NULL if not available. Input must be a point. This method implements the SQL/MM specification.
- ST_MLineFromText - Return a specified ST_MultiLineString value from WKT representation. 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 give, it defaults to -1. 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 give, it defaults to -1. This method implements the SQL/MM specification. SQL-MM 3: 9.6.4
- ST_NumGeometries - If geometry is a GEOMETRYCOLLECTION (or MULTI*) return the number of geometries, otherwise return NULL. This method implements the SQL/MM specification. SQL-MM 3: 9.1.4

- ST_NumInteriorRing - Return the number of interior rings of the first polygon in the geometry. Synonym to ST_NumInteriorRings. This method implements the SQL/MM specification. SQL-MM 3: 8.2.5
- ST_NumInteriorRings - Return the number of interior rings of the first polygon in the geometry. This will work with both POLYGON and MULTIPOLYGON types but only looks at the first polygon. Return NULL if there is no polygon in the geometry. This method implements the SQL/MM specification. SQL-MM 3: 8.2.5
- ST_NumPoints - Return the number of points in an ST_LineString or ST_CircularString value. This method implements the SQL/MM specification. SQL-MM 3: 7.2.4
- ST_OrderingEquals - Returns true if the given geometries represent the same geometry and points are in the same directional order. This method implements the SQL/MM specification. SQL-MM 3: 5.1.43
- ST_Overlaps - Returns TRUE if the Geometries share space, are of the same dimension, but are not completely contained by each other. This method implements the SQL/MM specification. SQL-MM 3: 5.1.32
- ST_Perimeter - Return the length measurement of the boundary of an ST_Surface or ST_MultiSurface value. (Polygon, Multipolygon) This method implements the SQL/MM specification. SQL-MM 3: 8.1.3, 9.5.4
- ST_Point - Returns an ST_Point with the given coordinate values. OGC alias for ST_MakePoint. This method implements the SQL/MM specification. SQL-MM 3: 6.1.2
- ST_PointFromText - Makes a point Geometry from WKT with the given SRID. If SRID is not given, it defaults to unknown. This method implements the SQL/MM specification. SQL-MM 3: 6.1.8
- ST_PointFromWKB - Makes a geometry from WKB with the given SRID This method implements the SQL/MM specification. SQL-MM 3: 6.1.9
- ST_PointN - Return the Nth point in the first linestring or circular linestring in the geometry. Return NULL if there is no linestring in the geometry. This method implements the SQL/MM specification. SQL-MM 3: 7.2.5, 7.3.5
- ST_PointOnSurface - Returns a POINT guaranteed to lie on the surface. This method implements the SQL/MM specification. SQL-MM 3: 8.1.5, 9.5.6. According to the specs, ST_PointOnSurface works for surface geometries (POLYGONs, MULTIPOLYGONS, CURVED POLYGONS). So PostGIS seems to be extending what the spec allows here. Most databases Oracle,DB II, ESRI SDE seem to only support this function for surfaces. SQL Server 2008 like PostGIS supports for all common geometries.
- ST_Polygon - Returns a polygon built from the specified linestring and SRID. This method implements the SQL/MM specification. SQL-MM 3: 8.3.2
- ST_PolygonFromText - Makes a Geometry from WKT with the given SRID. If SRID is not give, it defaults to -1. This method implements the SQL/MM specification. SQL-MM 3: 8.3.6
- ST_Relate - Returns true if this Geometry is spatially related to anotherGeometry, by testing for intersections between the Interior, Boundary and Exterior of the two geometries as specified by the values in the intersectionMatrixPattern. If no intersectionMatrixPattern is passed in, then returns the maximum intersectionMatrixPattern that relates the 2 geometries. This method implements the SQL/MM specification. SQL-MM 3: 5.1.25
- ST_SRID - Returns the spatial reference identifier for the ST_Geometry as defined in spatial_ref_sys table. This method implements the SQL/MM specification. SQL-MM 3: 5.1.5
- ST_StartPoint - Returns the first point of a LINESTRING geometry as a POINT. This method implements the SQL/MM specification. SQL-MM 3: 7.1.3
- ST_SymDifference - Returns a geometry that represents the portions of A and B that do not intersect. It is called a symmetric difference because ST_SymDifference(A,B) = ST_SymDifference(B,A). This method implements the SQL/MM specification. SQL-MM 3: 5.1.21
- ST_Touches - Returns TRUE if the geometries have at least one point in common, but their interiors do not intersect. This method implements the SQL/MM specification. SQL-MM 3: 5.1.28
- ST_Transform - Returns a new geometry with its coordinates transformed to the SRID referenced by the integer parameter. This method implements the SQL/MM specification. SQL-MM 3: 5.1.6

- ST_Union - Returns a geometry that represents the point set union of the Geometries. This method implements the SQL/MM specification. SQL-MM 3: 5.1.19 the z-index (elevation) when polygons are involved.
- ST_WKBToSQL - Return a specified ST_Geometry value from Well-Known Binary representation (WKB). This is an alias name for ST_GeomFromWKB that takes no srid This method implements the SQL/MM specification. SQL-MM 3: 5.1.36
- ST_WKTToSQL - Return a specified ST_Geometry value from Well-Known Text representation (WKT). This is an alias name for ST_GeomFromText This method implements the SQL/MM specification. SQL-MM 3: 5.1.34
- ST_Within - Returns true if the geometry A is completely inside geometry B This method implements the SQL/MM specification. SQL-MM 3: 5.1.30
- ST_X - Return the X coordinate of the point, or NULL if not available. Input must be a point. This method implements the SQL/MM specification. SQL-MM 3: 6.1.3
- ST_Y - Return the Y coordinate of the point, or NULL if not available. Input must be a point. This method implements the SQL/MM specification. SQL-MM 3: 6.1.4
- ST_Z - Return the Z coordinate of the point, or NULL if not available. Input must be a point. This method implements the SQL/MM specification.

8.3 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 tranform - (favoring UTM, Lambert Azimuthal (North/South), and falling back on mercator in worst case scenario)

- ST_Area - Returns the area of the surface if it is a polygon or multi-polygon. For "geometry" type area is in SRID units. For "geography" area is in square meters.
- ST_AsBinary - Return the Well-Known Binary (WKB) representation of the geometry/geography without SRID meta data.
- ST_AsGML - Return the geometry as a GML version 2 or 3 element.
- ST_AsGeoJSON - Return the geometry as a GeoJSON element.
- ST_AsKML - Return the geometry as a KML element. Several variants. Default version=2, default precision=15
- ST_AsSVG - Returns a Geometry in SVG path data given a geometry or geography object.
- ST_AsText - Return the Well-Known Text (WKT) representation of the geometry/geography without SRID metadata.
- ST_Buffer - (T) For geometry: Returns a geometry that represents all points whose distance from this Geometry is less than or equal to distance. Calculations are in the Spatial Reference System of this Geometry. For geography: Uses a planar transform wrapper. Introduced in 1.5 support for different end cap and mitre settings to control shape. buffer_style options: quad_segs=#,endcap=round|flat|square,join=round|mitre|bevel,mitre_limit=#.#
- ST_CoveredBy - Returns 1 (TRUE) if no point in Geometry/Geography A is outside Geometry/Geography B
- ST_Covers - Returns 1 (TRUE) if no point in Geometry B is outside Geometry A. For geography: if geography point B is not outside Polygon Geography A
- ST_DWithin - Returns true if the geometries are within the specified distance of one another. For geometry units are in those of spatial reference and For geography units are in meters and measurement is defaulted to use_spheroid=true (measure around spheroid), for faster check, use_spheroid=false to measure along sphere.

- ST_Distance - For geometry type Returns the 2-dimensional cartesian minimum distance (based on spatial ref) between two geometries in projected units. For geography type defaults to return spheroidal minimum distance between two geographies in meters.
- ST_GeogFromText - Return a specified geography value from Well-Known Text representation or extended (WKT).
- ST_GeogFromWKB - Creates a geography instance from a Well-Known Binary geometry representation (WKB) or extended Well Known Binary (EWKB).
- ST_GeographyFromText - Return a specified geography value from Well-Known Text representation or extended (WKT).
- = - Returns TRUE if A's bounding box is the same as B's (uses float4 boxes).
- && - Returns TRUE if A's bounding box overlaps B's.
- ST_Intersection - (T) Returns a geometry that represents the shared portion of geomA and geomB. The geography implementation does a transform to geometry to do the intersection and then transform back to WGS84.
- ST_Intersects - Returns TRUE if the Geometries/Geography "spatially intersect" - (share any portion of space) and FALSE if they don't (they are Disjoint). For geography -- tolerance is 0.00001 meters (so any points that close are considered to intersect)
- ST_Length - Returns the 2d length of the geometry if it is a linestring or multilinestring. geometry are in units of spatial reference and geography are in meters (default spheroid)

8.4 PostGIS Geometry Dump Functions

The functions given below are PostGIS functions that take as input or return as output a set of or single geometry_dump data type object.

- ST_Dump - Returns a set of geometry_dump (geom,path) rows, that make up a geometry g1.
- ST_DumpPoints - Returns a set of geometry_dump (geom,path) rows of all points that make up a geometry.
- ST_DumpRings - Returns a set of geometry_dump rows, representing the exterior and interior rings of a polygon.

8.5 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, box3d, box3d_extent

- Box2D - Returns a BOX2D representing the maximum extents of the geometry.
- Box3D - Returns a BOX3D representing the maximum extents of the geometry.
- ST_Estimated_Extent - Return the 'estimated' extent of the given spatial table. The estimated is taken from the geometry column's statistics. The current schema will be used if not specified.
- ST_Expand - Returns bounding box expanded in all directions from the bounding box of the input geometry. Uses double-precision
- ST_Extent - an aggregate function that returns the bounding box that bounds rows of geometries.
- ST_Extent3D - an aggregate function that returns the box3D bounding box that bounds rows of geometries.
- ST_MakeBox2D - Creates a BOX2D defined by the given point geometries.
- ST_MakeBox3D - Creates a BOX3D defined by the given 3d point geometries.
- ST_XMax - Returns X maxima of a bounding box 2d or 3d or a geometry.

- ST_XMin - Returns X minima of a bounding box 2d or 3d or a geometry.
- ST_YMax - Returns Y maxima of a bounding box 2d or 3d or a geometry.
- ST_YMin - Returns Y minima of a bounding box 2d or 3d or a geometry.
- ST_ZMax - Returns Z minima of a bounding box 2d or 3d or a geometry.
- ST_ZMin - Returns Z minima of a bounding box 2d or 3d or a geometry.

8.6 PostGIS Functions that support 3D

The functions given below are PostGIS functions that do not throw away the Z-Index.

- AddGeometryColumn - Adds a geometry column to an existing table of attributes.
- Box3D - Returns a BOX3D representing the maximum extents of the geometry.
- DropGeometryColumn - Removes a geometry column from a spatial table.
- ST_Accum - Aggregate. Constructs an array of geometries.
- ST_AddMeasure - Return a derived geometry with measure elements linearly interpolated between the start and end points. If the geometry has no measure dimension, one is added. If the geometry has a measure dimension, it is over-written with new values. Only LINESTRINGS and MULTILINESTRINGS are supported.
- ST_AddPoint - Adds a point to a LineString before point <position> (0-based index).
- ST_Affine - Applies a 3d affine transformation to the geometry to do things like translate, rotate, scale in one step.
- ST_AsEWKB - Return the Well-Known Binary (WKB) representation of the geometry with SRID meta data.
- ST_AsEWKT - Return the Well-Known Text (WKT) representation of the geometry with SRID meta data.
- ST_AsGML - Return the geometry as a GML version 2 or 3 element.
- ST_AsGeoJSON - Return the geometry as a GeoJSON element.
- ST_AsHEXEWKB - Returns a Geometry in HEXEWKB format (as text) using either little-endian (NDR) or big-endian (XDR) encoding.
- ST_AsKML - Return the geometry as a KML element. Several variants. Default version=2, default precision=15
- ST_Boundary - Returns the closure of the combinatorial boundary of this Geometry.
- ST_Collect - Return a specified ST_Geometry value from a collection of other geometries.
- ST_ConvexHull - The convex hull of a geometry represents the minimum convex geometry that encloses all geometries within the set.
- ST_CoordDim - Return the coordinate dimension of the ST_Geometry value.
- ST_CurveToLine - Converts a CIRCULARSTRING/CURVEDPOLYGON to a LINESTRING/POLYGON
- ST_Difference - Returns a geometry that represents that part of geometry A that does not intersect with geometry B.
- ST_Dump - Returns a set of geometry_dump (geom,path) rows, that make up a geometry g1.
- ST_DumpPoints - Returns a set of geometry_dump (geom,path) rows of all points that make up a geometry.
- ST_DumpRings - Returns a set of geometry_dump rows, representing the exterior and interior rings of a polygon.
- ST_EndPoint - Returns the last point of a LINESTRING geometry as a POINT.
- ST_Extent3D - an aggregate function that returns the box3D bounding box that bounds rows of geometries.

- ST_ExteriorRing - Returns a line string representing the exterior ring of the POLYGON geometry. Return NULL if the geometry is not a polygon. Will not work with MULTIPOLYGON
- ST_ForceRHR - Forces the orientation of the vertices in a polygon to follow the Right-Hand-Rule.
- ST_Force_3D - Forces the geometries into XYZ mode. This is an alias for ST_Force_3DZ.
- ST_Force_3DZ - Forces the geometries into XYZ mode. This is a synonym for ST_Force_3D.
- ST_Force_4D - Forces the geometries into XYZM mode.
- ST_Force_Collection - Converts the geometry into a GEOMETRYCOLLECTION.
- ST_GeomFromEWKB - Return a specified ST_Geometry value from Extended Well-Known Binary representation (EWKB).
- ST_GeomFromEWKT - Return a specified ST_Geometry value from Extended Well-Known Text representation (EWKT).
- ST_GeomFromGML - Takes as input GML representation of geometry and outputs a PostGIS geometry object
- ST_GeomFromKML - Takes as input KML representation of geometry and outputs a PostGIS geometry object
- ST_GeometryN - Return the 1-based Nth geometry if the geometry is a GEOMETRYCOLLECTION, MULTIPOINT, MULTILINESTRING, MULTICURVE or MULTIPOLYGON. Otherwise, return NULL.
- ST_HasArc - Returns true if a geometry or geometry collection contains a circular string
- ST_InteriorRingN - Return the Nth interior linestring ring of the polygon geometry. Return NULL if the geometry is not a polygon or the given N is out of range.
- ST_IsClosed - Returns TRUE if the LINESTRING's start and end points are coincident.
- ST_IsSimple - Returns (TRUE) if this Geometry has no anomalous geometric points, such as self intersection or self tangency.
- ST_Length3D - Returns the 3-dimensional or 2-dimensional length of the geometry if it is a linestring or multi-linestring.
- ST_Length3D_Spheroid - Calculates the length of a geometry on an ellipsoid, taking the elevation into account. This is just an alias for ST_Length_Spheroid.
- ST_Length_Spheroid - Calculates the 2D or 3D length of a linestring/multilinestring on an ellipsoid. This is useful if the coordinates of the geometry are in longitude/latitude and a length is desired without reprojection.
- ST_LineFromMultiPoint - Creates a LineString from a MultiPoint geometry.
- ST_LineToCurve - Converts a LINESTRING/POLYGON to a CIRCULARSTRING, CURVED POLYGON
- ST_Line_Interpolate_Point - Returns a point interpolated along a line. Second argument is a float8 between 0 and 1 representing fraction of total length of linestring the point has to be located.
- ST_Line_Substring - Return a linestring being a substring of the input one starting and ending at the given fractions of total 2d length. Second and third arguments are float8 values between 0 and 1.
- ST_LocateBetweenElevations - Return a derived geometry (collection) value with elements that intersect the specified range of elevations inclusively. Only 3D, 4D LINESTRINGS and MULTILINESTRINGS are supported.
- ST_M - Return the M coordinate of the point, or NULL if not available. Input must be a point.
- ST_MakeBox3D - Creates a BOX3D defined by the given 3d point geometries.
- ST_MakeLine - Creates a Linestring from point geometries.
- ST_MakePoint - Creates a 2D,3DZ or 4D point geometry.
- ST_MakePolygon - Creates a Polygon formed by the given shell. Input geometries must be closed LINESTRINGS.
- ST_MemUnion - Same as ST_Union, only memory-friendly (uses less memory and more processor time).
- ST_Mem_Size - Returns the amount of space (in bytes) the geometry takes.

- ST_NDims - Returns coordinate dimension of the geometry as a small int. Values are: 2,3 or 4.
- ST_NPoints - Return the number of points (vertexes) in a geometry.
- ST_NRings - If the geometry is a polygon or multi-polygon returns the number of rings.
- ST_Perimeter3D - Returns the 3-dimensional perimeter of the geometry, if it is a polygon or multi-polygon.
- ST_PointFromWKB - Makes a geometry from WKB with the given SRID
- ST_PointN - Return the Nth point in the first linestring or circular linestring in the geometry. Return NULL if there is no linestring in the geometry.
- ST_PointOnSurface - Returns a POINT guaranteed to lie on the surface.
- ST_Polygon - Returns a polygon built from the specified linestring and SRID.
- ST_RemovePoint - Removes point from a linestring. Offset is 0-based.
- ST_Rotate - This is a synonym for ST_RotateZ
- ST_RotateX - Rotate a geometry rotRadians about the X axis.
- ST_RotateY - Rotate a geometry rotRadians about the Y axis.
- ST_RotateZ - Rotate a geometry rotRadians about the Z axis.
- ST_Scale - Scales the geometry to a new size by multiplying the ordinates with the parameters. Ie: ST_Scale(geom, Xfactor, Yfactor, Zfactor).
- ST_SetPoint - Replace point N of linestring with given point. Index is 0-based.
- ST_Shift_Longitude - Reads every point/vertex in every component of every feature in a geometry, and if the longitude coordinate is <0, adds 360 to it. The result would be a 0-360 version of the data to be plotted in a 180 centric map
- ST_SnapToGrid - Snap all points of the input geometry to the grid defined by its origin and cell size. Remove consecutive points falling on the same cell, eventually returning NULL if output points are not enough to define a geometry of the given type. Collapsed geometries in a collection are stripped from it. Useful for reducing precision.
- ST_StartPoint - Returns the first point of a LINESTRING geometry as a POINT.
- ST_Summary - Returns a text summary of the contents of the ST_Geometry.
- ST_SymDifference - Returns a geometry that represents the portions of A and B that do not intersect. It is called a symmetric difference because ST_SymDifference(A,B) = ST_SymDifference(B,A).
- ST_TransScale - Translates the geometry using the deltaX and deltaY args, then scales it using the XFactor, YFactor args, working in 2D only.
- ST_Translate - Translates the geometry to a new location using the numeric parameters as offsets. Ie: ST_Translate(geom, X, Y) or ST_Translate(geom, X, Y,Z).
- ST_X - Return the X coordinate of the point, or NULL if not available. Input must be a point.
- ST_XMax - Returns X maxima of a bounding box 2d or 3d or a geometry.
- ST_XMin - Returns X minima of a bounding box 2d or 3d or a geometry.
- ST_Y - Return the Y coordinate of the point, or NULL if not available. Input must be a point.
- ST_YMax - Returns Y maxima of a bounding box 2d or 3d or a geometry.
- ST_YMin - Returns Y minima of a bounding box 2d or 3d or a geometry.
- ST_Z - Return the Z coordinate of the point, or NULL if not available. Input must be a point.
- ST_ZMax - Returns Z minima of a bounding box 2d or 3d or a geometry.

- ST_ZMin - Returns Z minima of a bounding box 2d or 3d or a geometry.
- ST_Zmflag - Returns ZM (dimension semantic) flag of the geometries as a small int. Values are: 0=2d, 1=3dm, 2=3dz, 3=4d.
- UpdateGeometrySRID - Updates the SRID of all features in a geometry column, geometry_columns metadata and srid table constraint

8.7 PostGIS Curved Geometry Support Functions

The functions given below are PostGIS functions that can use CIRCULARSTRING, CURVEDPOLYGON, and other curved geometry types

- AddGeometryColumn - Adds a geometry column to an existing table of attributes.
- Box2D - Returns a BOX2D representing the maximum extents of the geometry.
- Box3D - Returns a BOX3D representing the maximum extents of the geometry.
- DropGeometryColumn - Removes a geometry column from a spatial table.
- GeometryType - Returns the type of the geometry as a string. Eg: 'LINESTRING', 'POLYGON', 'MULTIPOINT', etc.
- 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_Accum - Aggregate. Constructs an array of geometries.
- ST_Affine - Applies a 3d affine transformation to the geometry to do things like translate, rotate, scale in one step.
- ST_AsBinary - Return the Well-Known Binary (WKB) representation of the geometry/geography without SRID meta data.
- ST_AsEWKB - Return the Well-Known Binary (WKB) representation of the geometry with SRID meta data.
- ST_AsEWKT - Return the Well-Known Text (WKT) representation of the geometry with SRID meta data.
- ST_AsHEXEWKB - Returns a Geometry in HEXEWKB format (as text) using either little-endian (NDR) or big-endian (XDR) encoding.
- ST_AsText - Return the Well-Known Text (WKT) representation of the geometry/geography without SRID metadata.
- ST_Collect - Return a specified ST_Geometry value from a collection of other geometries.
- ST_CoordDim - Return the coordinate dimension of the ST_Geometry value.
- ST_CurveToLine - Converts a CIRCULARSTRING/CURVEDPOLYGON to a LINESTRING/POLYGON
- ST_Dump - Returns a set of geometry_dump (geom,path) rows, that make up a geometry g1.
- ST_Estimated_Extent - Return the 'estimated' extent of the given spatial table. The estimated is taken from the geometry column's statistics. The current schema will be used if not specified.
- ST_Extent3D - an aggregate function that returns the box3D bounding box that bounds rows of geometries.
- ST_Force_2D - Forces the geometries into a "2-dimensional mode" so that all output representations will only have the X and Y coordinates.
- ST_Force_3D - Forces the geometries into XYZ mode. This is an alias for ST_Force_3DZ.
- ST_Force_3DM - Forces the geometries into XYM mode.
- ST_Force_3DZ - Forces the geometries into XYZ mode. This is a synonym for ST_Force_3D.

- ST_Force_4D - Forces the geometries into XYZM mode.
- ST_Force_Collection - Converts the geometry into a GEOMETRYCOLLECTION.
- ST_GeoHash - Return a GeoHash representation (geohash.org) of the geometry.
- ST_GeogFromWKB - Creates a geography instance from a Well-Known Binary geometry representation (WKB) or extended Well Known Binary (EWKB).
- ST_GeomFromEWKB - Return a specified ST_Geometry value from Extended Well-Known Binary representation (EWKB).
- ST_GeomFromEWKT - Return a specified ST_Geometry value from Extended Well-Known Text representation (EWKT).
- ST_GeomFromText - Return a specified ST_Geometry value from Well-Known Text representation (WKT).
- ST_GeomFromWKB - Creates a geometry instance from a Well-Known Binary geometry representation (WKB) and optional SRID.
- ST_GeometryN - Return the 1-based Nth geometry if the geometry is a GEOMETRYCOLLECTION, MULTIPOINT, MULTILINESTRING, MULTICURVE or MULTIPOLYGON. Otherwise, return NULL.
- = - Returns TRUE if A's bounding box is the same as B's (uses float4 boxes).
- &<| - Returns TRUE if A's bounding box overlaps or is below B's.
- && - Returns TRUE if A's bounding box overlaps B's.
- ST_HasArc - Returns true if a geometry or geometry collection contains a circular string
- ST_IsClosed - Returns TRUE if the LINESTRING's start and end points are coincident.
- ST_IsEmpty - Returns true if this Geometry is an empty geometry . If true, then this Geometry represents the empty point set - i.e. GEOMETRYCOLLECTION(EMPTY).
- ST_LineToCurve - Converts a LINESTRING/POLYGON to a CIRCULARSTRING, CURVED POLYGON
- ST_Mem_Size - Returns the amount of space (in bytes) the geometry takes.
- ST_NPoints - Return the number of points (vertexes) in a geometry.
- ST_NRings - If the geometry is a polygon or multi-polygon returns the number of rings.
- ST_PointFromWKB - Makes a geometry from WKB with the given SRID
- ST_PointN - Return the Nth point in the first linestring or circular linestring in the geometry. Return NULL if there is no linestring in the geometry.
- ST_Rotate - This is a synonym for ST_RotateZ
- ST_RotateZ - Rotate a geometry rotRadians about the Z axis.
- ST_SRID - Returns the spatial reference identifier for the ST_Geometry as defined in spatial_ref_sys table.
- ST_Scale - Scales the geometry to a new size by multiplying the ordinates with the parameters. Ie: ST_Scale(geom, Xfactor, Yfactor, Zfactor).
- ST_SetSRID - Sets the SRID on a geometry to a particular integer value.
- ST_TransScale - Translates the geometry using the deltaX and deltaY args, then scales it using the XFactor, YFactor args, working in 2D only.
- ST_Transform - Returns a new geometry with its coordinates transformed to the SRID referenced by the integer parameter.
- ST_Translate - Translates the geometry to a new location using the numeric parameters as offsets. Ie: ST_Translate(geom, X, Y) or ST_Translate(geom, X, Y,Z).
- ST_XMax - Returns X maxima of a bounding box 2d or 3d or a geometry.

- ST_XMin - Returns X minima of a bounding box 2d or 3d or a geometry.
- ST_YMax - Returns Y maxima of a bounding box 2d or 3d or a geometry.
- ST_YMin - Returns Y minima of a bounding box 2d or 3d or a geometry.
- ST_ZMax - Returns Z minima of a bounding box 2d or 3d or a geometry.
- ST_ZMin - Returns Z minima of a bounding box 2d or 3d or a geometry.
- ST_Zmflag - Returns ZM (dimension semantic) flag of the geometries as a small int. Values are: 0=2d, 1=3dm, 2=3dz, 3=4d.
- UpdateGeometrySRID - Updates the SRID of all features in a geometry column, geometry_columns metadata and srid table constraint

8.8 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.

Function	geometry	geography	3D (2.5D)	Curves	SQL MM
Box2D	✓			✓	
Box3D	✓		✓	✓	
Find_SRID					
GeometryType	✓			✓	
ST_Accum	✓		✓	✓	
ST_AddMeasure	✓		✓		
ST_AddPoint	✓		✓		
ST_Affine	✓		✓	✓	
ST_Area	✓	✓			✓
ST_AsBinary	✓	✓		✓	✓
ST_AsEWKB	✓		✓	✓	
ST_AsEWKT	✓		✓	✓	
ST_AsGML	✓	✓	✓		
ST_AsGeoJSON	✓	✓	✓		
ST_AsHEXEWKB	✓		✓	✓	
ST_AsKML	✓	✓	✓		

Function	geometry	geography	3D (2.5D)	Curves	SQL MM
ST_AsSVG	✓	✓			
ST_AsText	✓	✓		✓	✓
ST_Azimuth	✓				
ST_BdMPolyFromText	✓				
ST_BdPolyFromText	✓				
ST_Boundary	✓		✓		✓
ST_Buffer	✓	😁			✓
ST_BuildArea	✓				
ST_Centroid	✓				✓
ST_ClosestPoint	✓				
ST_Collect	✓		✓	✓	
ST_CollectionExtract	✓				
ST_Contains	✓				✓
ST_ContainsProperly	✓				
ST_ConvexHull	✓		✓		✓
ST_CoordDim	✓		✓	✓	✓
ST_CoveredBy	✓	✓			
ST_Covers	✓	✓			
ST_Crosses	✓				✓
ST_CurveToLine	✓		✓	✓	✓
ST_DFullyWithin	✓				
ST_DWithin	✓	✓			
ST_Difference	✓		✓		✓
ST_Dimension	✓				✓
ST_Disjoint	✓				✓
ST_Distance	✓	✓			✓
ST_Distance_Sphere	✓				
ST_Distance_Spheroid	✓				
ST_Dump	✓		✓	✓	
ST_DumpPoints	✓		✓		
ST_DumpRings	✓		✓		
ST_EndPoint	✓		✓		✓
ST_Envelope	✓				✓
ST_Equals	✓				✓

Function	geometry	geography	3D (2.5D)	Curves	SQL MM
ST_Estimated_Extent	☑			✓	
ST_Expand	✓				
ST_Extent	✓				
ST_Extent3D	✓		✓	✓	
ST_ExteriorRing	✓		✓		✓
ST_ForceRHR	✓		✓		
ST_Force_2D	✓			✓	
ST_Force_3D	✓		✓	✓	
ST_Force_3DM	✓			✓	
ST_Force_3DZ	✓		✓	✓	
ST_Force_4D	✓		✓	✓	
ST_Force_Collection	✓		✓	✓	
ST_GMLToSQL	✓				✓
ST_GeoHash	✓			✓	
ST_GeogFromText		✓			
ST_GeogFromWKB		✓		✓	
ST_GeographyFromText		✓			
ST_GeomCollFromText	✓				✓
ST_GeomFromEWKB	✓		✓	✓	
ST_GeomFromEWKT	✓		✓	✓	
ST_GeomFromGML	✓		✓		
ST_GeomFromKML	✓		✓		
ST_GeomFromText	✓			✓	✓
ST_GeomFromWKB	✓			✓	✓
ST_GeometryFromText	✓				✓
ST_GeometryN	✓		✓	✓	✓
ST_GeometryType	✓				✓
|>>	✓				
<<|	✓				
~	✓				
@	✓				
=	✓	✓		✓	
<<	✓				
|&>	✓				

Function	geometry	geography	3D (2.5D)	Curves	SQL MM
&<|	✓			✓	
&&	✓	✓		✓	
&<	✓				
&>	✓				
>>	✓				
~=	✓				
ST_HasArc	✓		✓	✓	
ST_HausdorffDistance	✓				
ST_InteriorRingN	✓		✓		✓
ST_Intersection	✓	😁			✓
ST_Intersects	✓	✓			✓
ST_IsClosed	✓		✓	✓	✓
ST_IsEmpty	✓			✓	✓
ST_IsRing	✓				✓
ST_IsSimple	✓		✓		✓
ST_IsValid	✓				✓
ST_IsValidReason	✓				
ST_Length	✓	✓			✓
ST_Length2D	✓				
ST_Length2D_Spheroid	✓				
ST_Length3D	✓		✓		
ST_Length3D_Spheroid	✓		✓		
ST_Length_Spheroid	✓		✓		
ST_LineCrossingDirection	✓				
ST_LineFromMultiPoint	✓		✓		
ST_LineFromText	✓				✓
ST_LineFromWKB	✓				✓
ST_LineMerge	✓				
ST_LineToCurve	✓		✓	✓	
ST_Line_Interpolate_Point	✓		✓		
ST_Line_Locate_Point	✓				
ST_Line_Substring	✓		✓		
ST_LinestringFromWKB	✓				✓
ST_LocateBetweenElevatio	✓		✓		

Function	geometry	geography	3D (2.5D)	Curves	SQL MM
ST_Locate_Along_Measure	✓				
ST_Locate_Between_Meas	✓				
ST_LongestLine	✓				
ST_M	✓		✓		✓
ST_MLineFromText	✓				✓
ST_MPointFromText	✓				✓
ST_MPolyFromText	✓				✓
ST_MakeBox2D	✓				
ST_MakeBox3D	✓		✓		
ST_MakeEnvelope	✓				
ST_MakeLine	✓		✓		
ST_MakePoint	✓		✓		
ST_MakePointM	✓				
ST_MakePolygon	✓		✓		
ST_MaxDistance	✓				
ST_MemUnion	✓		✓		
ST_Mem_Size	✓		✓	✓	
ST_MinimumBoundingCirc	✓				
ST_Multi	✓				
ST_NDims	✓		✓		
ST_NPoints	✓		✓	✓	
ST_NRings	✓		✓	✓	
ST_NumGeometries	✓				✓
ST_NumInteriorRing	✓				✓
ST_NumInteriorRings	✓				✓
ST_NumPoints	✓				✓
ST_OrderingEquals	✓				✓
ST_Overlaps	✓				✓
ST_Perimeter	✓				✓
ST_Perimeter2D	✓				
ST_Perimeter3D	✓		✓		
ST_Point	✓				✓
ST_PointFromText	✓				✓
ST_PointFromWKB	✓		✓	✓	✓

Function	geometry	geography	3D (2.5D)	Curves	SQL MM
ST_PointN	✓		✓	✓	✓
ST_PointOnSurface	✓		✓		✓
ST_Point_Inside_Circle	✓				
ST_Polygon	✓		✓		✓
ST_PolygonFromText	✓				✓
ST_Polygonize	✓				
ST_Relate	✓				✓
ST_RemovePoint	✓		✓		
ST_Reverse	✓				
ST_Rotate	✓		✓	✓	
ST_RotateX	✓		✓		
ST_RotateY	✓		✓		
ST_RotateZ	✓		✓	✓	
ST_SRID	✓			✓	✓
ST_Scale	✓		✓	✓	
ST_Segmentize	✓				
ST_SetPoint	✓		✓		
ST_SetSRID	✓			✓	
ST_Shift_Longitude	✓		✓		
ST_ShortestLine	✓				
ST_Simplify	✓				
ST_SimplifyPreserveTopolo	✓				
ST_SnapToGrid	✓		✓		
ST_StartPoint	✓		✓		✓
ST_Summary	✓		✓		
ST_SymDifference	✓		✓		✓
ST_Touches	✓				✓
ST_TransScale	✓		✓	✓	
ST_Transform	✓			✓	✓
ST_Translate	✓		✓	✓	
ST_Union	✓				✓
ST_WKBToSQL	✓				✓
ST_WKTToSQL	✓				✓
ST_Within	✓				✓

Function	geometry	geography	3D (2.5D)	Curves	SQL MM
ST_X	✓		✓		✓
ST_XMax	☑		✓	✓	
ST_XMin	☑		✓	✓	
ST_Y	✓		✓		✓
ST_YMax	☑		✓	✓	
ST_YMin	☑		✓	✓	
ST_Z	✓		✓		✓
ST_ZMax	☑		✓	✓	
ST_ZMin	☑		✓	✓	
ST_Zmflag	✓		✓	✓	

8.9 New PostGIS Functions

8.9.1 PostGIS Functions new, behavior changed, or enhanced in 1.5

The functions given below are PostGIS functions that were introduced or enhanced in this major release.

- PostGIS_LibXML_Version - Availability: 1.5 Returns the version number of the libxml2 library.
- ST_AddMeasure - Availability: 1.5.0 Return a derived geometry with measure elements linearly interpolated between the start and end points. If the geometry has no measure dimension, one is added. If the geometry has a measure dimension, it is over-written with new values. Only LINESTRINGS and MULTILINESTRINGS are supported.
- ST_AsBinary - Availability: 1.5.0 geography support was introduced. Return the Well-Known Binary (WKB) representation of the geometry/geography without SRID meta data.
- ST_AsGeoJSON - Availability: 1.5.0 geography support was introduced. Return the geometry as a GeoJSON element.
- ST_AsText - Availability: 1.5 - support for geography was introduced. Return the Well-Known Text (WKT) representation of the geometry/geography without SRID metadata.
- 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. - requires GEOS >= 3.2 to take advantage of advanced geometry functionality. (T) For geometry: Returns a geometry that represents all points whose distance from this Geometry is less than or equal to distance. Calculations are in the Spatial Reference System of this Geometry. For geography: Uses a planar transform wrapper. Introduced in 1.5 support for different end cap and mitre settings to control shape. buffer_style options: quad_segs=#,endcap=round|flat|square,join=round|mitre|bevel,
- ST_ClosestPoint - Availability: 1.5.0 Returns the 2-dimensional point on g1 that is closest to g2. This is the first point of the shortest line.
- ST_CollectionExtract - Availability: 1.5.0 Given a GEOMETRYCOLLECTION, returns a MULTI* geometry consisting only of the specified type. Sub-geometries that are not the specified type are ignored. If there are no sub-geometries of the right type, an EMPTY collection will be returned. Only points, lines and polygons are supported. Type numbers are 1 == POINT, 2 == LINESTRING, 3 == POLYGON.
- ST_Covers - Availability: 1.5 - support for geography was introduced. Returns 1 (TRUE) if no point in Geometry B is outside Geometry A. For geography: if geography point B is not outside Polygon Geography A
- ST_DFullyWithin - Availability: 1.5.0 Returns true if all of the geometries are within the specified distance of one another

- ST_DWithin - Availability: 1.5.0 support for geography was introduced Returns true if the geometries are within the specified distance of one another. For geometry units are in those of spatial reference and For geography units are in meters and measurement is defaulted to use_spheroid=true (measure around spheroid), for faster check, use_spheroid=false to measure along sphere.
- ST_Distance - Availability: 1.5.0 geography support was introduced in 1.5. Speed improvements for planar to better handle large or many vertex geometries For geometry type Returns the 2-dimensional cartesian minimum distance (based on spatial ref) between two geometries in projected units. For geography type defaults to return spheroidal minimum distance between two geographies in meters.
- ST_Distance_Sphere - Availability: 1.5 - support for other geometry types besides points was introduced. Prior versions only work with points. Returns minimum distance in meters between two lon/lat geometries. Uses a spherical earth and radius of 6370986 meters. Faster than ST_Distance_Spheroid, but less accurate. PostGIS versions prior to 1.5 only implemented for points.
- ST_Distance_Spheroid - Availability: 1.5 - support for other geometry types besides points was introduced. Prior versions only work with points. Returns the minimum distance between two lon/lat geometries given a particular spheroid. PostGIS versions prior to 1.5 only support points.
- ST_DumpPoints - Availability: 1.5.0 Returns a set of geometry_dump (geom,path) rows of all points that make up a geometry.
- ST_Envelope - Availability: 1.5.0 behavior changed to output double precision instead of float4 Returns a geometry representing the double precision (float8) bounding box of the supplied geometry.
- ST_GMLToSQL - Availability: 1.5 Return a specified ST_Geometry value from GML representation. This is an alias name for ST_GeomFromGML
- ST_GeomFromGML - Availability: 1.5 Takes as input GML representation of geometry and outputs a PostGIS geometry object
- ST_GeomFromKML - Availability: 1.5 Takes as input KML representation of geometry and outputs a PostGIS geometry object
- && - Availability: 1.5.0 support for geography was introduced. Returns TRUE if A's bounding box overlaps B's.
- ~= - Availability: 1.5.0 changed behavior Returns TRUE if A's bounding box is the same as B's.
- ST_HausdorffDistance - Availability: 1.5.0 - requires GEOS >= 3.2.0 Returns the Hausdorff distance between two geometries. Basically a measure of how similar or dissimilar 2 geometries are. Units are in the units of the spatial reference system of the geometries.
- ST_Intersection - Availability: 1.5 support for geography data type was introduced. (T) Returns a geometry that represents the shared portion of geomA and geomB. The geography implementation does a transform to geometry to do the intersection and then transform back to WGS84.
- ST_Intersects - Availability: 1.5 support for geography was introduced. Returns TRUE if the Geometries/Geography "spatially intersect" - (share any portion of space) and FALSE if they don't (they are Disjoint). For geography -- tolerance is 0.00001 meters (so any points that close are considered to intersect)
- ST_Length - Availability: 1.5.0 geography support was introduced in 1.5. Returns the 2d length of the geometry if it is a linestring or multilinestring. geometry are in units of spatial reference and geography are in meters (default spheroid)
- ST_LongestLine - Availability: 1.5.0 Returns the 2-dimensional longest line points of two geometries. The function will only return the first longest line if more than one, that the function finds. The line returned will always start in g1 and end in g2. The length of the line this function returns will always be the same as st_maxdistance returns for g1 and g2.
- ST_MakeEnvelope - Availability: 1.5 Creates a rectangular Polygon formed from the given minimums and maximums. Input values must be in SRS specified by the SRID.
- ST_MaxDistance - Availability: 1.5.0 Returns the 2-dimensional largest distance between two geometries in projected units.
- ST_ShortestLine - Availability: 1.5.0 Returns the 2-dimensional shortest line between two geometries

8.9.2 PostGIS Functions new, behavior changed, or enhanced in 1.4

The functions given below are PostGIS functions that were introduced or enhanced in the 1.4 release.

- Populate_Geometry_Columns - Ensures geometry columns have appropriate spatial constraints and exist in the geometry_columns table. Availability: 1.4.0
- ST_AsSVG - Returns a Geometry in SVG path data given a geometry or geography object. Availability: 1.2.2 . Availability: 1.4.0 Changed in PostGIS 1.4.0 to include L command in absolute path to conform to http://www.w3.org/TR/SVG/paths.html#PathData
- ST_Collect - Return a specified ST_Geometry value from a collection of other geometries. Availability: 1.4.0 - ST_Collect(geomarray) was introduced. ST_Collect was enhanced to handle more geometries faster.
- ST_ContainsProperly - Returns true if B intersects the interior of A but not the boundary (or exterior). A does not contain properly itself, but does contain itself. Availability: 1.4.0 - requires GEOS >= 3.1.0.
- ST_Extent - an aggregate function that returns the bounding box that bounds rows of geometries. Availability: 1.4.0 As of 1.4.0 now returns a box3d_extent instead of box2d object.
- ST_GeoHash - Return a GeoHash representation (geohash.org) of the geometry. Availability: 1.4.0
- ST_IsValidReason - Returns text stating if a geometry is valid or not and if not valid, a reason why. Availability: 1.4 - requires GEOS >= 3.1.0.
- ST_LineCrossingDirection - Given 2 linestrings, returns a number between -3 and 3 denoting what kind of crossing behavior. 0 is no crossing. Availability: 1.4
- ST_LocateBetweenElevations - Return a derived geometry (collection) value with elements that intersect the specified range of elevations inclusively. Only 3D, 4D LINESTRINGS and MULTILINESTRINGS are supported. Availability: 1.4.0
- ST_MakeLine - Creates a Linestring from point geometries. Availability: 1.4.0 - ST_MakeLine(geomarray) was introduced. ST_MakeLine aggregate functions was enhanced to handle more points faster.
- ST_MinimumBoundingCircle - Returns the smallest circle polygon that can fully contain a geometry. Default uses 48 segments per quarter circle. Availability: 1.4.0 - requires GEOS
- ST_Union - Returns a geometry that represents the point set union of the Geometries. Availability: 1.4.0 - ST_Union was enhanced. ST_Union(geomarray) was introduced and also faster aggregate collection in PostgreSQL. If you are using GEOS 3.1.0+ ST_Union will use the faster Cascaded Union algorithm described in http://blog.cleverelephant.ca/2009/01/must-faster-unions-in-postgis-14.html

8.9.3 PostGIS Functions new in 1.3

The functions given below are PostGIS functions that were introduced in the 1.3 release.

- ST_AsGeoJSON - Return the geometry as a GeoJSON element. Availability: 1.3.4
- ST_SimplifyPreserveTopology - Returns a "simplified" version of the given geometry using the Douglas-Peucker algorithm. Will avoid creating derived geometries (polygons in particular) that are invalid. Availability: 1.3.3

Chapter 9

Reporting Problems

9.1 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.

9.2 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 Subversion patch is definitely preferred. This is a four step process on Unix (assuming you already have Subversion installed):

1. Check out a copy of PostGIS' Subversion trunk. On Unix, type:

   svn checkout http://svn.osgeo.org/postgis/trunk/

   This will be stored in the directory ./trunk

2. Make your changes to the documentation with your favorite text editor. On Unix, type (for example):

   vim trunk/doc/postgis.xml

   Note that the documentation is written in SGML 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:

   svn diff trunk/doc/postgis.xml > doc.patch

4. Attach the patch to a new issue in bug tracker.

Appendix A

Appendix

A.1 Release 1.5.8

Release date: 2012/11/15

This is a bug fix release, addressing issues that have been filed since the 1.5.7 release.

A.1.1 Bug Fixes

#2048, ST_Within and ST_CoveredBy producing bad results

#2095, proj4 cache corrupted by projection error

A.2 Release 1.5.7

Release date: 2012/10/31

This is a bug fix release, addressing issues that have been filed since the 1.5.6 release.

A.2.1 Bug Fixes

#2071, Add PgSQL 9.2 support

A.3 Release 1.5.6

Release date: 2012/10/30

This is a bug fix release, addressing issues that have been filed since the 1.5.5 release.

A.3.1 Bug Fixes

#547, ST_Contains memory problems, the remake

#1957, ST_Distance to a one-point LineString returns NULL

#1936, ST_GeomFromGML on CurvePolygon causes server crash

#1953, Segfault on GEOS calls with empty polygon

#1976, Geography point-in-ring code overhauled for more reliability

#2071, Add PgSQL 9.2 support

A.4 Release 1.5.5

Release date: 2012/07/20

This is a bug fix release, addressing issues that have been filed since the 1.5.4 release.

A.4.1 Bug Fixes

#1825, containsproperly fix in prepared geometry.

#1832, Crash when updating GIST index on geography column

#1865, don't strip comments COPY data from dumps on restore

A.5 Release 1.5.4

Release date: 2012/05/06

This is a bug fix release, addressing issues that have been filed since the 1.5.3 release.

A.5.1 Bug Fixes

#547, ST_Contains memory problems (Sandro Santilli)

#621, Problem finding intersections with geography (Paul Ramsey)

#627, PostGIS/PostgreSQL process die on invalid geometry (Paul Ramsey)

#810, Increase accuracy of area calculation (Paul Ramsey)

#852, improve spatial predicates robustness (Sandro Santilli, Nicklas Avén)

#877, ST_Estimated_Extent returns NULL on empty tables (Sandro Santilli)

#1028, ST_AsSVG kills whole postgres server when fails (Paul Ramsey)

#1056, Fix boxes of arcs and circle stroking code (Paul Ramsey)

#1121, populate_geometry_columns using deprecated functions (Regin Obe, Paul Ramsey)

#1135, improve testsuite predictability (Andreas 'ads' Scherbaum)

#1146, images generator crashes (bronaugh)

#1170, North Pole intersection fails (Paul Ramsey)

#1179, ST_AsText crash with bad value (kjurka)

#1184, honour DESTDIR in documentation Makefile (Bryce L Nordgren)

#1227, server crash on invalid GML

#1252, SRID appearing in WKT (Paul Ramsey)

#1264, st_dwithin(g, g, 0) doesn't work (Paul Ramsey)

#1344, allow exporting tables with invalid geometries (Sandro Santilli)

#1389, wrong proj4text for SRID 31300 and 31370 (Paul Ramsey)

#1406, shp2pgsql crashes when loading into geography (Sandro Santilli)

#1595, fixed SRID redundancy in ST_Line_SubString (Sandro Santilli)

#1596, check SRID in UpdateGeometrySRID (Mike Toews, Sandro Santilli)

#1602, fix ST_Polygonize to retain Z (Sandro Santilli)

#1697, fix crash with EMPTY entries in GiST index (Paul Ramsey)

#1772, fix ST_Line_Locate_Point with collapsed input (Sandro Santilli)

#1799, Protect ST_Segmentize from max_length=0 (Sandro Santilli)

Alter parameter order in 900913 (Paul Ramsey)

Support builds with "gmake" (Greg Troxel)

A.6 Release 1.5.3

Release date: 2011/06/25

This is a bug fix release, addressing issues that have been filed since the 1.5.2 release.

A.6.1 Bug Fixes

#1007, ST_IsValid crash fix requires GEOS 3.3.0+ or 3.2.3+ (Sandro Santilli, reported by Birgit Laggner)

#940, support for PostgreSQL 9.1 beta 1 (Regina Obe, Paul Ramsey, patch submitted by stl)

#845, ST_Intersects precision error (Sandro Santilli, Nicklas Avén) Reported by cdestigter

#884, Unstable results with ST_Within, ST_Intersects (Chris Hodgson)

#779, shp2pgsql -S option seems to fail on points (Jeff Adams)

#666, ST_DumpPoints is not null safe (Regina Obe)

#631, Update NZ projections for grid transformation support (jpalmer)

#630, Peculiar Null treatment in arrays in ST_Collect (Chris Hodgson) Reported by David Bitner

#624, Memory leak in ST_GeogFromText (ryang, Paul Ramsey)

#609, Bad source code in manual section 5.2 Java Clients (simoc, Regina Obe)

#604, shp2pgsql usage touchups (Mike Toews, Paul Ramsey)

#573 ST_Union fails on a group of linestrings Not a PostGIS bug, fixed in GEOS 3.3.0

#457 ST_CollectionExtract returns non-requested type (Nicklas Avén, Paul Ramsey)

#441 ST_AsGeoJson Bbox on GeometryCollection error (Olivier Courtin)

#411 Ability to backup invalid geometries (Sando Santilli) Reported by Regione Toscana

#409 ST_AsSVG - degraded (Olivier Courtin) Reported by Sdikiy

#373 Documentation syntax error in hard upgrade (Paul Ramsey) Reported by psvensso

A.7 Release 1.5.2

Release date: 2010/09/27

This is a bug fix release, addressing issues that have been filed since the 1.5.1 release.

A.7.1 Bug Fixes

Loader: fix handling of empty (0-verticed) geometries in shapefiles. (Sandro Santilli)

#536, Geography ST_Intersects, ST_Covers, ST_CoveredBy and Geometry ST_Equals not using spatial index (Regina Obe, Nicklas Aven)

#573, Improvement to ST_Contains geography (Paul Ramsey)

Loader: Add support for command-q shutdown in Mac GTK build (Paul Ramsey)

#393, Loader: Add temporary patch for large DBF files (Maxime Guillaud, Paul Ramsey)

#507, Fix wrong OGC URN in GeoJSON and GML output (Olivier Courtin)

spatial_ref_sys.sql Add datum conversion for projection SRID 3021 (Paul Ramsey)

Geography - remove crash for case when all geographies are out of the estimate (Paul Ramsey)

#469, Fix for array_aggregation error (Greg Stark, Paul Ramsey)

#532, Temporary geography tables showing up in other user sessions (Paul Ramsey)

#562, ST_Dwithin errors for large geographies (Paul Ramsey)

#513, shape loading GUI tries to make spatial index when loading DBF only mode (Paul Ramsey)

#527, shape loading GUI should always append log messages (Mark Cave-Ayland)

#504, shp2pgsql should rename xmin/xmax fields (Sandro Santilli)

#458, postgis_comments being installed in contrib instead of version folder (Mark Cave-Ayland)

#474, Analyzing a table with geography column crashes server (Paul Ramsey)

#581, LWGEOM-expand produces inconsistent results (Mark Cave-Ayland)

#513, Add dbf filter to shp2pgsql-gui and allow uploading dbf only (Paul Ramsey)

Fix further build issues against PostgreSQL 9.0 (Mark Cave-Ayland)

#572, Password whitespace for Shape File

#603, shp2pgsql: "-w" produces invalid WKT for MULTI* objects. (Mark Cave-Ayland)

A.8 Release 1.5.1

Release date: 2010/03/11

This is a bug fix release, addressing issues that have been filed since the 1.4.1 release.

A.8.1 Bug Fixes

#410, update embedded bbox when applying ST_SetPoint, ST_AddPoint ST_RemovePoint to a linestring (Paul Ramsey)

#411, allow dumping tables with invalid geometries (Sandro Santilli, for Regione Toscana-SIGTA)

#414, include geography_columns view when running upgrade scripts (Paul Ramsey)

#419, allow support for multilinestring in ST_Line_Substring (Paul Ramsey, for Lidwala Consulting Engineers)

#421, fix computed string length in ST_AsGML() (Olivier Courtin)

#441, fix GML generation with heterogeneous collections (Olivier Courtin)

#443, incorrect coordinate reversal in GML 3 generation (Olivier Courtin)

#450, #451, wrong area calculation for geography features that cross the date line (Paul Ramsey)

Ensure support for upcoming 9.0 PgSQL release (Paul Ramsey)

A.9 Release 1.5.0

Release date: 2010/02/04

This release provides support for geographic coordinates (lat/lon) via a new GEOGRAPHY type. Also performance enhancements, new input format support (GML,KML) and general upkeep.

A.9.1 API Stability

The public API of PostGIS will not change during minor (0.0.X) releases.

The definition of the =~ operator has changed from an exact geometric equality check to a bounding box equality check.

A.9.2 Compatibility

GEOS, Proj4, and LibXML2 are now mandatory dependencies

The library versions below are the minimum requirements for PostGIS 1.5

PostgreSQL 8.3 and higher on all platforms

GEOS 3.1 and higher only (GEOS 3.2+ to take advantage of all features)

LibXML2 2.5+ related to new ST_GeomFromGML/KML functionality

Proj4 4.5 and higher only

A.9.3 New Features

Section 8.9.1

Added Hausdorff distance calculations (#209) (Vincent Picavet)

Added parameters argument to ST_Buffer operation to support one-sided buffering and other buffering styles (Sandro Santilli)

Addition of other Distance related visualization and analysis functions (Nicklas Aven)

- ST_ClosestPoint
- ST_DFullyWithin
- ST_LongestLine
- ST_MaxDistance
- ST_ShortestLine

ST_DumpPoints (Maxime van Noppen)

KML, GML input via ST_GeomFromGML and ST_GeomFromKML (Olivier Courtin)

Extract homogeneous collection with ST_CollectionExtract (Paul Ramsey)

Add measure values to an existing linestring with ST_AddMeasure (Paul Ramsey)

History table implementation in utils (George Silva)

Geography type and supporting functions

- Spherical algorithms (Dave Skea)
- Object/index implementation (Paul Ramsey)
- Selectivity implementation (Mark Cave-Ayland)
- Serializations to KML, GML and JSON (Olivier Courtin)
- ST_Area, ST_Distance, ST_DWithin, ST_GeogFromText, ST_GeogFromWKB, ST_Intersects, ST_Covers, ST_Buffer (Paul Ramsey)

A.9.4 Enhancements

Performance improvements to ST_Distance (Nicklas Aven)

Documentation updates and improvements (Regina Obe, Kevin Neufeld)

Testing and quality control (Regina Obe)

PostGIS 1.5 support PostgreSQL 8.5 trunk (Guillaume Lelarge)

Win32 support and improvement of core shp2pgsql-gui (Mark Cave-Ayland)

In place 'make check' support (Paul Ramsey)

A.9.5 Bug fixes

http://trac.osgeo.org/postgis/query?status=closed&milestone=postgis+1.5.0&order=priority

A.10 Release 1.4.0

Release date: 2009/07/24

This release provides performance enhancements, improved internal structures and testing, new features, and upgraded documentation.

A.10.1 API Stability

As of the 1.4 release series, the public API of PostGIS will not change during minor releases.

A.10.2 Compatibility

The versions below are the *minimum* requirements for PostGIS 1.4

PostgreSQL 8.2 and higher on all platforms

GEOS 3.0 and higher only

PROJ4 4.5 and higher only

A.10.3 New Features

ST_Union() uses high-speed cascaded union when compiled against GEOS 3.1+ (Paul Ramsey)

ST_ContainsProperly() requires GEOS 3.1+

ST_Intersects(), ST_Contains(), ST_Within() use high-speed cached prepared geometry against GEOS 3.1+ (Paul Ramsey)

Vastly improved documentation and reference manual (Regina Obe & Kevin Neufeld)

Figures and diagram examples in the reference manual (Kevin Neufeld)

ST_IsValidReason() returns readable explanations for validity failures (Paul Ramsey)

ST_GeoHash() returns a geohash.org signature for geometries (Paul Ramsey)

GTK+ multi-platform GUI for shape file loading (Paul Ramsey)

ST_LineCrossingDirection() returns crossing directions (Paul Ramsey)

ST_LocateBetweenElevations() returns sub-string based on Z-ordinate. (Paul Ramsey)

Geometry parser returns explicit error message about location of syntax errors (Mark Cave-Ayland)

ST_AsGeoJSON() return JSON formatted geometry (Olivier Courtin)

Populate_Geometry_Columns() -- automatically add records to geometry_columns for TABLES and VIEWS (Kevin Neufeld)

ST_MinimumBoundingCircle() -- returns the smallest circle polygon that can encompass a geometry (Bruce Rindahl)

A.10.4 Enhancements

Core geometry system moved into independent library, liblwgeom. (Mark Cave-Ayland)

New build system uses PostgreSQL "pgxs" build bootstrapper. (Mark Cave-Ayland)

Debugging framework formalized and simplified. (Mark Cave-Ayland)

All build-time #defines generated at configure time and placed in headers for easier cross-platform support (Mark Cave-Ayland)

Logging framework formalized and simplified (Mark Cave-Ayland)

Expanded and more stable support for CIRCULARSTRING, COMPOUNDCURVE and CURVEPOLYGON, better parsing, wider support in functions (Mark Leslie & Mark Cave-Ayland)

Improved support for OpenSolaris builds (Paul Ramsey)

Improved support for MSVC builds (Mateusz Loskot)

Updated KML support (Olivier Courtin)

Unit testing framework for liblwgeom (Paul Ramsey)

New testing framework to comprehensively exercise every PostGIS function (Regine Obe)

Performance improvements to all geometry aggregate functions (Paul Ramsey)

Support for the upcoming PostgreSQL 8.4 (Mark Cave-Ayland, Talha Bin Rizwan)

Shp2pgsql and pgsql2shp re-worked to depend on the common parsing/unparsing code in liblwgeom (Mark Cave-Ayland)

Use of PDF DbLatex to build PDF docs and preliminary instructions for build (Jean David Techer)

Automated User documentation build (PDF and HTML) and Developer Doxygen Documentation (Kevin Neufeld)

Automated build of document images using ImageMagick from WKT geometry text files (Kevin Neufeld)

More attractive CSS for HTML documentation (Dane Springmeyer)

A.10.5 Bug fixes

http://trac.osgeo.org/postgis/query?status=closed&milestone=postgis+1.4.0&order=priority

A.11 Release 1.3.6

Release date: 2009/05/04

This release adds support for PostgreSQL 8.4, exporting prj files from the database with shape data, some crash fixes for shp2pgsql, and several small bug fixes in the handling of "curve" types, logical error importing dbf only files, improved error handling of AddGeometryColumns.

A.12 Release 1.3.5

Release date: 2008/12/15

This release is a bug fix release to address a failure in ST_Force_Collection and related functions that critically affects using Mapserver with LINE layers.

A.13 Release 1.3.4

Release date: 2008/11/24

This release adds support for GeoJSON output, building with PostgreSQL 8.4, improves documentation quality and output aesthetics, adds function-level SQL documentation, and improves performance for some spatial predicates (point-in-polygon tests).

Bug fixes include removal of crashers in handling circular strings for many functions, some memory leaks removed, a linear referencing failure for measures on vertices, and more. See the NEWS file for details.

A.14 Release 1.3.3

Release date: 2008/04/12

This release fixes bugs shp2pgsql, adds enhancements to SVG and KML support, adds a ST_SimplifyPreserveTopology function, makes the build more sensitive to GEOS versions, and fixes a handful of severe but rare failure cases.

A.15 Release 1.3.2

Release date: 2007/12/01

This release fixes bugs in ST_EndPoint() and ST_Envelope, improves support for JDBC building and OS/X, and adds better support for GML output with ST_AsGML(), including GML3 output.

A.16 Release 1.3.1

Release date: 2007/08/13

This release fixes some oversights in the previous release around version numbering, documentation, and tagging.

A.17 Release 1.3.0

Release date: 2007/08/09

This release provides performance enhancements to the relational functions, adds new relational functions and begins the migration of our function names to the SQL-MM convention, using the spatial type (SP) prefix.

A.17.1 Added Functionality

JDBC: Added Hibernate Dialect (thanks to Norman Barker)

Added ST_Covers and ST_CoveredBy relational functions. Description and justification of these functions can be found at http://lin-ear-th-inking.blogspot.com/2007/06/subtleties-of-ogc-covers-spatial.html

Added ST_DWithin relational function.

A.17.2 Performance Enhancements

Added cached and indexed point-in-polygon short-circuits for the functions ST_Contains, ST_Intersects, ST_Within and ST_Disjoint

Added inline index support for relational functions (except ST_Disjoint)

A.17.3 Other Changes

Extended curved geometry support into the geometry accessor and some processing functions

Began migration of functions to the SQL-MM naming convention; using a spatial type (ST) prefix.

Added initial support for PostgreSQL 8.3

A.18 Release 1.2.1

Release date: 2007/01/11

This release provides bug fixes in PostgreSQL 8.2 support and some small performance enhancements.

A.18.1 Changes

Fixed point-in-polygon shortcut bug in Within().

Fixed PostgreSQL 8.2 NULL handling for indexes.

Updated RPM spec files.

Added short-circuit for Transform() in no-op case.

JDBC: Fixed JTS handling for multi-dimensional geometries (thanks to Thomas Marti for hint and partial patch). Additionally, now JavaDoc is compiled and packaged. Fixed classpath problems with GCJ. Fixed pgjdbc 8.2 compatibility, losing support for jdk 1.3 and older.

A.19 Release 1.2.0

Release date: 2006/12/08

This release provides type definitions along with serialization/deserialization capabilities for SQL-MM defined curved geometries, as well as performance enhancements.

A.19.1 Changes

Added curved geometry type support for serialization/deserialization

Added point-in-polygon shortcircuit to the Contains and Within functions to improve performance for these cases.

A.20 Release 1.1.6

Release date: 2006/11/02

This is a bugfix release, in particular fixing a critical error with GEOS interface in 64bit systems. Includes an updated of the SRS parameters and an improvement in reprojections (take Z in consideration). Upgrade is *encouraged*.

A.20.1 Upgrading

If you are upgrading from release 1.0.3 or later follow the soft upgrade procedure.

If you are upgrading from a release *between 1.0.0RC6 and 1.0.2* (inclusive) and really want a live upgrade read the upgrade section of the 1.0.3 release notes chapter.

Upgrade from any release prior to 1.0.0RC6 requires an hard upgrade.

A.20.2 Bug fixes

fixed CAPI change that broke 64-bit platforms

loader/dumper: fixed regression tests and usage output

Fixed setSRID() bug in JDBC, thanks to Thomas Marti

A.20.3 Other changes

use Z ordinate in reprojections

spatial_ref_sys.sql updated to EPSG 6.11.1

Simplified Version.config infrastructure to use a single pack of version variables for everything.

Include the Version.config in loader/dumper USAGE messages

Replace hand-made, fragile JDBC version parser with Properties

A.21 Release 1.1.5

Release date: 2006/10/13

This is an bugfix release, including a critical segfault on win32. Upgrade is *encouraged*.

A.21.1 Upgrading

If you are upgrading from release 1.0.3 or later follow the soft upgrade procedure.

If you are upgrading from a release *between 1.0.0RC6 and 1.0.2* (inclusive) and really want a live upgrade read the upgrade section of the 1.0.3 release notes chapter.

Upgrade from any release prior to 1.0.0RC6 requires an hard upgrade.

A.21.2 Bug fixes

Fixed MingW link error that was causing pgsql2shp to segfault on Win32 when compiled for PostgreSQL 8.2

fixed nullpointer Exception in Geometry.equals() method in Java

Added EJB3Spatial.odt to fulfill the GPL requirement of distributing the "preferred form of modification"

Removed obsolete synchronization from JDBC Jts code.

Updated heavily outdated README files for shp2pgsql/pgsql2shp by merging them with the manpages.

Fixed version tag in jdbc code that still said "1.1.3" in the "1.1.4" release.

A.21.3 New Features

Added -S option for non-multi geometries to shp2pgsql

A.22 Release 1.1.4

Release date: 2006/09/27

This is an bugfix release including some improvements in the Java interface. Upgrade is *encouraged*.

A.22.1 Upgrading

If you are upgrading from release 1.0.3 or later follow the soft upgrade procedure.

If you are upgrading from a release *between 1.0.0RC6 and 1.0.2* (inclusive) and really want a live upgrade read the upgrade section of the 1.0.3 release notes chapter.

Upgrade from any release prior to 1.0.0RC6 requires an hard upgrade.

A.22.2 Bug fixes

Fixed support for PostgreSQL 8.2

Fixed bug in collect() function discarding SRID of input

Added SRID match check in MakeBox2d and MakeBox3d

Fixed regress tests to pass with GEOS-3.0.0

Improved pgsql2shp run concurrency.

A.22.3 Java changes

reworked JTS support to reflect new upstream JTS developers' attitude to SRID handling. Simplifies code and drops build depend on GNU trove.

Added EJB2 support generously donated by the "Geodetix s.r.l. Company" http://www.geodetix.it/

Added EJB3 tutorial / examples donated by Norman Barker <nbarker@ittvis.com>

Reorganized java directory layout a little.

A.23 Release 1.1.3

Release date: 2006/06/30

This is an bugfix release including also some new functionalities (most notably long transaction support) and portability enhancements. Upgrade is *encouraged.*

A.23.1 Upgrading

If you are upgrading from release 1.0.3 or later follow the soft upgrade procedure.

If you are upgrading from a release *between 1.0.0RC6 and 1.0.2* (inclusive) and really want a live upgrade read the upgrade section of the 1.0.3 release notes chapter.

Upgrade from any release prior to 1.0.0RC6 requires an hard upgrade.

A.23.2 Bug fixes / correctness

BUGFIX in distance(poly,poly) giving wrong results.

BUGFIX in pgsql2shp successful return code.

BUGFIX in shp2pgsql handling of MultiLine WKT.

BUGFIX in affine() failing to update bounding box.

WKT parser: forbidden construction of multigeometries with EMPTY elements (still supported for GEOMETRYCOLLECTION).

A.23.3 New functionalities

NEW Long Transactions support.

NEW DumpRings() function.

NEW AsHEXEWKB(geom, XDR|NDR) function.

A.23.4 JDBC changes

Improved regression tests: MultiPoint and scientific ordinates

Fixed some minor bugs in jdbc code

Added proper accessor functions for all fields in preparation of making those fields private later

A.23.5 Other changes

NEW regress test support for loader/dumper.

Added --with-proj-libdir and --with-geos-libdir configure switches.

Support for build Tru64 build.

Use Jade for generating documentation.

Don't link pgsql2shp to more libs then required.

Initial support for PostgreSQL 8.2.

A.24 Release 1.1.2

Release date: 2006/03/30

This is an bugfix release including some new functions and portability enhancements. Upgrade is *encouraged*.

A.24.1 Upgrading

If you are upgrading from release 1.0.3 or later follow the soft upgrade procedure.

If you are upgrading from a release *between 1.0.0RC6 and 1.0.2* (inclusive) and really want a live upgrade read the upgrade section of the 1.0.3 release notes chapter.

Upgrade from any release prior to 1.0.0RC6 requires an hard upgrade.

A.24.2 Bug fixes

BUGFIX in SnapToGrid() computation of output bounding box

BUGFIX in EnforceRHR()

jdbc2 SRID handling fixes in JTS code

Fixed support for 64bit archs

A.24.3 New functionalities

Regress tests can now be run *before* postgis installation

New affine() matrix transformation functions

New rotate{,X,Y,Z}() function

Old translating and scaling functions now use affine() internally

Embedded access control in estimated_extent() for builds against pgsql >= 8.0.0

A.24.4 Other changes

More portable ./configure script

Changed ./run_test script to have more sane default behaviour

A.25 Release 1.1.1

Release date: 2006/01/23

This is an important Bugfix release, upgrade is *highly recommended*. Previous version contained a bug in postgis_restore.pl preventing hard upgrade procedure to complete and a bug in GEOS-2.2+ connector preventing GeometryCollection objects to be used in topological operations.

A.25.1 Upgrading

If you are upgrading from release 1.0.3 or later follow the soft upgrade procedure.

If you are upgrading from a release *between 1.0.0RC6 and 1.0.2* (inclusive) and really want a live upgrade read the upgrade section of the 1.0.3 release notes chapter.

Upgrade from any release prior to 1.0.0RC6 requires an hard upgrade.

A.25.2 Bug fixes

Fixed a premature exit in postgis_restore.pl

BUGFIX in geometrycollection handling of GEOS-CAPI connector

Solaris 2.7 and MingW support improvements

BUGFIX in line_locate_point()

Fixed handling of postgresql paths

BUGFIX in line_substring()

Added support for localized cluster in regress tester

A.25.3 New functionalities

New Z and M interpolation in line_substring()

New Z and M interpolation in line_interpolate_point()

added NumInteriorRing() alias due to OpenGIS ambiguity

A.26 Release 1.1.0

Release date: 2005/12/21

This is a Minor release, containing many improvements and new things. Most notably: build procedure greatly simplified; transform() performance drastically improved; more stable GEOS connectivity (CAPI support); lots of new functions; draft topology support.

It is *highly recommended* that you upgrade to GEOS-2.2.x before installing PostGIS, this will ensure future GEOS upgrades won't require a rebuild of the PostGIS library.

A.26.1 Credits

This release includes code from Mark Cave Ayland for caching of proj4 objects. Markus Schaber added many improvements in his JDBC2 code. Alex Bodnaru helped with PostgreSQL source dependency relief and provided Debian specfiles. Michael Fuhr tested new things on Solaris arch. David Techer and Gerald Fenoy helped testing GEOS C-API connector. Hartmut Tschauner provided code for the azimuth() function. Devrim GUNDUZ provided RPM specfiles. Carl Anderson helped with the new area building functions. See the credits section for more names.

A.26.2 Upgrading

If you are upgrading from release 1.0.3 or later you *DO NOT* need a dump/reload. Simply sourcing the new lwpostgis_upgrade.sql script in all your existing databases will work. See the soft upgrade chapter for more information.

If you are upgrading from a release *between 1.0.0RC6 and 1.0.2* (inclusive) and really want a live upgrade read the upgrade section of the 1.0.3 release notes chapter.

Upgrade from any release prior to 1.0.0RC6 requires an hard upgrade.

A.26.3 New functions

scale() and transscale() companion methods to translate()

line_substring()

line_locate_point()

M(point)

LineMerge(geometry)

shift_longitude(geometry)

azimuth(geometry)

locate_along_measure(geometry, float8)

locate_between_measures(geometry, float8, float8)

SnapToGrid by point offset (up to 4d support)

BuildArea(any_geometry)

OGC BdPolyFromText(linestring_wkt, srid)

OGC BdMPolyFromText(linestring_wkt, srid)

RemovePoint(linestring, offset)

ReplacePoint(linestring, offset, point)

A.26.4 Bug fixes

Fixed memory leak in polygonize()

Fixed bug in lwgeom_as_anytype cast functions

Fixed USE_GEOS, USE_PROJ and USE_STATS elements of postgis_version() output to always reflect library state.

A.26.5 Function semantic changes

SnapToGrid doesn't discard higher dimensions

Changed Z() function to return NULL if requested dimension is not available

A.26.6 Performance improvements

Much faster transform() function, caching proj4 objects

Removed automatic call to fix_geometry_columns() in AddGeometryColumns() and update_geometry_stats()

A.26.7 JDBC2 works

Makefile improvements

JTS support improvements

Improved regression test system

Basic consistency check method for geometry collections

Support for (Hex)(E)wkb

Autoprobing DriverWrapper for HexWKB / EWKT switching

fix compile problems in ValueSetter for ancient jdk releases.

fix EWKT constructors to accept SRID=4711; representation

added preliminary read-only support for java2d geometries

A.26.8 Other new things

Full autoconf-based configuration, with PostgreSQL source dependency relief

GEOS C-API support (2.2.0 and higher)

Initial support for topology modelling

Debian and RPM specfiles

New lwpostgis_upgrade.sql script

A.26.9 Other changes

JTS support improvements

Stricter mapping between DBF and SQL integer and string attributes

Wider and cleaner regression test suite

old jdbc code removed from release

obsoleted direct use of postgis_proc_upgrade.pl

scripts version unified with release version

A.27 Release 1.0.6

Release date: 2005/12/06

Contains a few bug fixes and improvements.

A.27.1 Upgrading

If you are upgrading from release 1.0.3 or later you *DO NOT* need a dump/reload.

If you are upgrading from a release *between 1.0.0RC6 and 1.0.2* (inclusive) and really want a live upgrade read the upgrade section of the 1.0.3 release notes chapter.

Upgrade from any release prior to 1.0.0RC6 requires an hard upgrade.

A.27.2 Bug fixes

Fixed palloc(0) call in collection deserializer (only gives problem with --enable-cassert)

Fixed bbox cache handling bugs

Fixed geom_accum(NULL, NULL) segfault

Fixed segfault in addPoint()

Fixed short-allocation in lwcollection_clone()

Fixed bug in segmentize()

Fixed bbox computation of SnapToGrid output

A.27.3 Improvements

Initial support for postgresql 8.2

Added missing SRID mismatch checks in GEOS ops

A.28 Release 1.0.5

Release date: 2005/11/25

Contains memory-alignment fixes in the library, a segfault fix in loader's handling of UTF8 attributes and a few improvements and cleanups.

Note
Return code of shp2pgsql changed from previous releases to conform to unix standards (return 0 on success).

A.28.1 Upgrading

If you are upgrading from release 1.0.3 or later you *DO NOT* need a dump/reload.

If you are upgrading from a release *between 1.0.0RC6 and 1.0.2* (inclusive) and really want a live upgrade read the upgrade section of the 1.0.3 release notes chapter.

Upgrade from any release prior to 1.0.0RC6 requires an hard upgrade.

A.28.2 Library changes

Fixed memory alignment problems

Fixed computation of null values fraction in analyzer

Fixed a small bug in the getPoint4d_p() low-level function

Speedup of serializer functions

Fixed a bug in force_3dm(), force_3dz() and force_4d()

A.28.3 Loader changes

Fixed return code of shp2pgsql

Fixed back-compatibility issue in loader (load of null shapefiles)

Fixed handling of trailing dots in dbf numerical attributes

Segfault fix in shp2pgsql (utf8 encoding)

A.28.4 Other changes

Schema aware postgis_proc_upgrade.pl, support for pgsql 7.2+

New "Reporting Bugs" chapter in manual

A.29 Release 1.0.4

Release date: 2005/09/09

Contains important bug fixes and a few improvements. In particular, it fixes a memory leak preventing successful build of GiST indexes for large spatial tables.

A.29.1 Upgrading

If you are upgrading from release 1.0.3 you *DO NOT* need a dump/reload.

If you are upgrading from a release *between 1.0.0RC6 and 1.0.2* (inclusive) and really want a live upgrade read the upgrade section of the 1.0.3 release notes chapter.

Upgrade from any release prior to 1.0.0RC6 requires an hard upgrade.

A.29.2 Bug fixes

Memory leak plugged in GiST indexing

Segfault fix in transform() handling of proj4 errors

Fixed some proj4 texts in spatial_ref_sys (missing +proj)

Loader: fixed string functions usage, reworked NULL objects check, fixed segfault on MULTILINESTRING input.

Fixed bug in MakeLine dimension handling

Fixed bug in translate() corrupting output bounding box

A.29.3 Improvements

Documentation improvements

More robust selectivity estimator

Minor speedup in distance()

Minor cleanups

GiST indexing cleanup

Looser syntax acceptance in box3d parser

A.30 Release 1.0.3

Release date: 2005/08/08

Contains some bug fixes - *including a severe one affecting correctness of stored geometries* - and a few improvements.

A.30.1 Upgrading

Due to a bug in a bounding box computation routine, the upgrade procedure requires special attention, as bounding boxes cached in the database could be incorrect.

An hard upgrade procedure (dump/reload) will force recomputation of all bounding boxes (not included in dumps). This is *required* if upgrading from releases prior to 1.0.0RC6.

If you are upgrading from versions 1.0.0RC6 or up, this release includes a perl script (utils/rebuild_bbox_caches.pl) to force recomputation of geometries' bounding boxes and invoke all operations required to propagate eventual changes in them (geometry statistics update, reindexing). Invoke the script after a make install (run with no args for syntax help). Optionally run utils/postgis_proc_upgrade.pl to refresh postgis procedures and functions signatures (see Soft upgrade).

A.30.2 Bug fixes

Severe bugfix in lwgeom's 2d bounding box computation

Bugfix in WKT (-w) POINT handling in loader

Bugfix in dumper on 64bit machines

Bugfix in dumper handling of user-defined queries

Bugfix in create_undef.pl script

A.30.3 Improvements

Small performance improvement in canonical input function

Minor cleanups in loader

Support for multibyte field names in loader

Improvement in the postgis_restore.pl script

New rebuild_bbox_caches.pl util script

A.31 Release 1.0.2

Release date: 2005/07/04

Contains a few bug fixes and improvements.

A.31.1 Upgrading

If you are upgrading from release 1.0.0RC6 or up you *DO NOT* need a dump/reload.

Upgrading from older releases requires a dump/reload. See the upgrading chapter for more informations.

A.31.2 Bug fixes

Fault tolerant btree ops

Memory leak plugged in pg_error

Rtree index fix

Cleaner build scripts (avoided mix of CFLAGS and CXXFLAGS)

A.31.3 Improvements

New index creation capabilities in loader (-I switch)

Initial support for postgresql 8.1dev

A.32 Release 1.0.1

Release date: 2005/05/24

Contains a few bug fixes and some improvements.

A.32.1 Upgrading

If you are upgrading from release 1.0.0RC6 or up you *DO NOT* need a dump/reload.

Upgrading from older releases requires a dump/reload. See the upgrading chapter for more informations.

A.32.2 Library changes

BUGFIX in 3d computation of length_spheroid()

BUGFIX in join selectivity estimator

A.32.3 Other changes/additions

BUGFIX in shp2pgsql escape functions

better support for concurrent postgis in multiple schemas

documentation fixes

jdbc2: compile with "-target 1.2 -source 1.2" by default

NEW -k switch for pgsql2shp

NEW support for custom createdb options in postgis_restore.pl

BUGFIX in pgsql2shp attribute names unicity enforcement

BUGFIX in Paris projections definitions

postgis_restore.pl cleanups

A.33 Release 1.0.0

Release date: 2005/04/19

Final 1.0.0 release. Contains a few bug fixes, some improvements in the loader (most notably support for older postgis versions), and more docs.

A.33.1 Upgrading

If you are upgrading from release 1.0.0RC6 you *DO NOT* need a dump/reload.

Upgrading from any other precedent release requires a dump/reload. See the upgrading chapter for more informations.

A.33.2 Library changes

BUGFIX in transform() releasing random memory address

BUGFIX in force_3dm() allocating less memory then required

BUGFIX in join selectivity estimator (defaults, leaks, tuplecount, sd)

A.33.3 Other changes/additions

BUGFIX in shp2pgsql escape of values starting with tab or single-quote

NEW manual pages for loader/dumper

NEW shp2pgsql support for old (HWGEOM) postgis versions

NEW -p (prepare) flag for shp2pgsql

NEW manual chapter about OGC compliancy enforcement

NEW autoconf support for JTS lib

BUGFIX in estimator testers (support for LWGEOM and schema parsing)

A.34 Release 1.0.0RC6

Release date: 2005/03/30

Sixth release candidate for 1.0.0. Contains a few bug fixes and cleanups.

A.34.1 Upgrading

You need a dump/reload to upgrade from precedent releases. See the upgrading chapter for more informations.

A.34.2 Library changes

BUGFIX in multi()

early return [when noop] from multi()

A.34.3 Scripts changes

dropped {x,y}{min,max}(box2d) functions

A.34.4 Other changes

BUGFIX in postgis_restore.pl scrip

BUGFIX in dumper's 64bit support

A.35 Release 1.0.0RC5

Release date: 2005/03/25

Fifth release candidate for 1.0.0. Contains a few bug fixes and a improvements.

A.35.1 Upgrading

If you are upgrading from release 1.0.0RC4 you *DO NOT* need a dump/reload.

Upgrading from any other precedent release requires a dump/reload. See the upgrading chapter for more informations.

A.35.2 Library changes

BUGFIX (segfaulting) in box3d computation (yes, another!).

BUGFIX (segfaulting) in estimated_extent().

A.35.3 Other changes

Small build scripts and utilities refinements.

Additional performance tips documented.

A.36 Release 1.0.0RC4

Release date: 2005/03/18

Fourth release candidate for 1.0.0. Contains bug fixes and a few improvements.

A.36.1 Upgrading

You need a dump/reload to upgrade from precedent releases. See the upgrading chapter for more informations.

A.36.2 Library changes

BUGFIX (segfaulting) in geom_accum().

BUGFIX in 64bit architectures support.

BUGFIX in box3d computation function with collections.

NEW subselects support in selectivity estimator.

Early return from force_collection.

Consistency check fix in SnapToGrid().

Box2d output changed back to 15 significant digits.

A.36.3 Scripts changes

NEW distance_sphere() function.

Changed get_proj4_from_srid implementation to use PL/PGSQL instead of SQL.

A.36.4 Other changes

BUGFIX in loader and dumper handling of MultiLine shapes

BUGFIX in loader, skipping all but first hole of polygons.

jdbc2: code cleanups, Makefile improvements

FLEX and YACC variables set *after* pgsql Makefile.global is included and only if the pgsql *stripped* version evaluates to the empty string

Added already generated parser in release

Build scripts refinements

improved version handling, central Version.config

improvements in postgis_restore.pl

A.37 Release 1.0.0RC3

Release date: 2005/02/24

Third release candidate for 1.0.0. Contains many bug fixes and improvements.

A.37.1 Upgrading

You need a dump/reload to upgrade from precedent releases. See the upgrading chapter for more informations.

A.37.2 Library changes

BUGFIX in transform(): missing SRID, better error handling.

BUGFIX in memory alignment handling

BUGFIX in force_collection() causing mapserver connector failures on simple (single) geometry types.

BUGFIX in GeometryFromText() missing to add a bbox cache.

reduced precision of box2d output.

prefixed DEBUG macros with PGIS_ to avoid clash with pgsql one

plugged a leak in GEOS2POSTGIS converter

Reduced memory usage by early releasing query-context palloced one.

A.37.3 Scripts changes

BUGFIX in 72 index bindings.

BUGFIX in probe_geometry_columns() to work with PG72 and support multiple geometry columns in a single table

NEW bool::text cast

Some functions made IMMUTABLE from STABLE, for performance improvement.

A.37.4 JDBC changes

jdbc2: small patches, box2d/3d tests, revised docs and license.

jdbc2: bug fix and testcase in for pgjdbc 8.0 type autoregistration

jdbc2: Removed use of jdk1.4 only features to enable build with older jdk releases.

jdbc2: Added support for building against pg72jdbc2.jar

jdbc2: updated and cleaned makefile

jdbc2: added BETA support for jts geometry classes

jdbc2: Skip known-to-fail tests against older PostGIS servers.

jdbc2: Fixed handling of measured geometries in EWKT.

A.37.5 Other changes

new performance tips chapter in manual

documentation updates: pgsql72 requirement, lwpostgis.sql

few changes in autoconf

BUILDDATE extraction made more portable

fixed spatial_ref_sys.sql to avoid vacuuming the whole database.

spatial_ref_sys: changed Paris entries to match the ones distributed with 0.x.

A.38 Release 1.0.0RC2

Release date: 2005/01/26

Second release candidate for 1.0.0 containing bug fixes and a few improvements.

A.38.1 Upgrading

You need a dump/reload to upgrade from precedent releases. See the upgrading chapter for more informations.

A.38.2 Library changes

BUGFIX in pointarray box3d computation

BUGFIX in distance_spheroid definition

BUGFIX in transform() missing to update bbox cache

NEW jdbc driver (jdbc2)

GEOMETRYCOLLECTION(EMPTY) syntax support for backward compatibility

Faster binary outputs

Stricter OGC WKB/WKT constructors

A.38.3 Scripts changes

More correct STABLE, IMMUTABLE, STRICT uses in lwpostgis.sql

stricter OGC WKB/WKT constructors

A.38.4 Other changes

Faster and more robust loader (both i18n and not)

Initial autoconf script

A.39 Release 1.0.0RC1

Release date: 2005/01/13

This is the first candidate of a major postgis release, with internal storage of postgis types redesigned to be smaller and faster on indexed queries.

A.39.1 Upgrading

You need a dump/reload to upgrade from precedent releases. See the upgrading chapter for more informations.

A.39.2 Changes

Faster canonical input parsing.

Lossless canonical output.

EWKB Canonical binary IO with PG>73.

Support for up to 4d coordinates, providing lossless shapefile->postgis->shapefile conversion.

New function: UpdateGeometrySRID(), AsGML(), SnapToGrid(), ForceRHR(), estimated_extent(), accum().

Vertical positioning indexed operators.

JOIN selectivity function.

More geometry constructors / editors.

PostGIS extension API.

UTF8 support in loader.