Manual PostGIS.

Nota:

He traducido el Manual PostGIS de **Paul Ramsey** para facilitar su uso a los usuarios de lengua castellana. Agradecería que me comunicasen las erratas y comentarios a mmm876@yahoo.es.

Manuel Martín Martín

Índice de contenido

PostGIS: Es una extensión al sistema de base de datos objeto-relacional PostgreSQL. Permite el uso de objetos *GIS*(Geographic information systems). PostGIS incluye soporte para índices GiST basados en R-Tree, y funciones básicas para el análisis de objetos GIS.

Esta creado por Refractions Research Inc, como un proyecto de investigación de tecnologías de bases de datos espaciales. Esta publicado bajo licencia GNU.

Con PostGIS podemos usar todos los objetos que aparecen en la especificación OpenGIS como puntos,lineas, polígonos, multilíneas, multipuntos, y colecciones geométricas.

1.Objetos GIS.

Los objetos GIS soportados por PostGIS son de características simples definidas por OpenGIS. Actualmente PostGIS soporta las características y el API de representación de la especificación OpenGIS pero no tiene varios de los operadores de comparación y convolución de esta especificación.

Ejemplos de la representación en modo texto:

- POINT(0 0 0)
- LINESTRING(0 0,1 1,1 2)
- 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))
- MULTIPOINT(0 0 0,1 2 1)
- MULTILINESTRING((0 0 0,1 1 0,1 2 1),(2 3 1,3 2 1,5 4 1))
- 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)))
- GEOMETRYCOLLECTION(POINT(2 3 9),LINESTRING((2 3 4,3 4 5))

En los ejemplos se pueden ver características con coordenadas de 2D y 3D(ambas son permitidas por PostGIS). Podemos usar las funciones force_2d() y force_3d() para convertir una característica a 3d o 2d.

1.1.Forma CANONICA vs ESTANDAR.

OpenGIS define dos formas de representar los objetos espaciales:

1. (WKT)Well-know text como los ejemplos anteriores.
2. (WKB)Well-know binary.

Las dos formas guardan información del tipo de objeto y sus coordenadas.

Además la especificación OpenGIS requiere que los objetos incluyan el identificador del sistema de referencia espacial(SRID).El SRID es requerido cuando insertamos un objeto espacial en la base de datos.

Ejemplo:

```
INSERT INTO SPATIALDATABASE(THE_GEOM,THE_NAME)
VALUES(GeometryFromText('POINT(-126.4  45.32)',312),'Un Lugar')
```

La función GeometryFromText requiere un numero SRID.

En PostgreSQL tenemos la representación en *forma canónica*, es una representación en modo texto. Esta representación es distinta al estándar openGIS.

Diferencias:

```
db=> SELECT AsText(geom) AS OGCGeom FROM thetable;
OGCGeom
------------------------------------------------
LINESTRING(-123.741378393049 48.9124018962261,-123.741587115639 48.9123981907507)
(1 row)

db=> SELECT geom AS PostGISGeom FROM thetable;
PostGISGeom
------------------------------------------------
SRID=123;LINESTRING(-123.741378393049 48.9124018962261,-123.741587115639 48.9123981907507)
(1 row)
```

Ilustración 1. Diferencias de representación entre PostgreSQL y OpenGIS.

2. Usar el estándar OpenGIS.

Las especificación para SQL de características simples de OpenGIS define tipos de objetos GIS estándar, los cuales son manipulados por funciones, y un conjunto de tablas de meta-datos.

Hay 2 tablas de meta-datos en la especificación OpenGIS:

2.1. SPATIAL_REF_SYS:

Contiene un identificador numérico y una descripción textual de el sistema de coordenadas espacial de la base de datos.

Como se define la tabla:

```
CREATE TALBE SPATIAL_REF_SYS(
SRID INTEGER NOT NULL PRIMARY KEY,
AUTH_NAME VARCHAR(256),
AUTH_SRID INTEGER,
SRTEXT VARCHAR(2048),
PROJ4TEXT VARCHAR(2048)
)
```

Las columnas de las tablas son la siguientes:

- SRID: Valor entero que identifica el sistema de referencia espacial.
- AUTH_NAME: El nombre del estándar para el sistema de referencia. Por ejemplo: EPSG.
- AUTH_SRID: El identificador según el estándar AUTH_NAME. En el ejemplo anterior es el id según EPSG.
- SRTEXT: Una *Well-know text* representación para el sistema de referencia espacial. Ejemplo: WKT para SRS.

```
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]
]
```

Para una lista de las representaciones WKT de EPSG ir a

http://www.opengis.org/techno/interop/EPSG2WKT.TXT.

- PROJ4TEXT: Proj4 es una librería que usa PostGIS para transformar coordenadas . Esta columna contiene una cadena con definición de las coordenadas de Proj4 para un SRID dado.

Ejemplo:

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

2.2. GEOMETRY_COLUMNS:

Definición:

```
CREATE TABLE GEOMETRY_COLUMNS(
F_TABLE_CATALOG VARCHAR(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
)
```

Descripción de cada una de las columnas:

- F_TABLE_CATALOG,F_TABLE_SCHEMA,F_TABLE_NAME.

  Distingue totalmente la tabla de características que contiene la columna geométrica.

- F_GEOMETRY_COLUMN

  Nombre de la columna geométrica en la tabla de características.

- COORD_DIMENSION

  Dimensión espacial de la columna(2D o 3D).

- SRID

  Es una clave foránea que referencia SPATIAL_REF_SYS.

- TYPE

  Tipo de objeto espacial. POINT, LINESTRING, POLYGON, MULTYPOINT, GEOMETRYCOLLECTION. Para un tipo heterogéneo debo usar el tipo GEOMETRY.

2.3. Crear una tabla espacial

Para crear una tabla con datos espaciales realizamos dos pasos:

1. Creamos una tabla no espacial.

```
Ejemplo: CREATE TABLE CALLES_GEOM(ID int4,NAME varchar(25))
```

2. Añadimos una columna(campo) espacial a la tabla usando la función *AddGeometryColumn* de OpenGIS.

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

```
Ejemplo:  SELECT AddGeometryColum('calles_db','calles_geom','geom',423, 'LINESTRING',2)
```

Ejemplos de creación de tablas con columnas geométricas:

- Parques:

  1) *CREATE TABLE PARQUES(PARQUE_ID int4,PARQUE_NOMBRE varchar(128),PARQUE_FECHA*

date);

2) *SELECT AddGeometryColumn('parque_db', 'parque', 'parque_geom', 128, 'MULTIPOLYGON' , 2) ;*

- Calles: El tipo de dato espacial es genérico(GEOMETRY).

  1)*CREATE TABLE CALLES(CALLE_ID int4,CALLE_NOMBRE varchar(128));*

  2)*SELECT AddGeometryColumn('calles_db','calles','calles_geom',-1,'GEOMETRY',3);*

3. Cargar datos GIS en la base de datos Espacial.

Hay dos formas de cargar datos en las tablas de nuestra base de datos. Usando el lenguaje SQL o usando un cargador de archivos de figuras.

1. Usando SQL.

   Si podemos convertir los datos que vamos ha insertar en un representación textual. Usar el formato SQL es una manera sencilla de insertar los datos en PostGIS. Podemos crear un archivo de texto lleno de sentencias INSERT y cargarlo con con SQL monitor.

   Ejemplo:

   calles.sql

```
BEGIN;
INSERT INTO CALLES_GEOM(ID,GEOM,NAME) VALUES (1, GeometryFromText
('LINESTRING(191232 243118,191108 243242)',-1),'Jeff Rd');
INSERT INTO CALLES_GEOM(ID,GEOM,NAME) VALUES (1, GeometryFromText
('LINESTRING(189141 244158,189265 244817)',-1),'Geordie Rd');
INSERT INTO CALLES_GEOM(ID,GEOM,NAME) VALUES (1, GeometryFromText
('LINESTRING(192783  228138,192612  229814)',-1),'Paul St');
INSERT INTO CALLES_GEOM(ID,GEOM,NAME) VALUES (1, GeometryFromText
('LINESTRING(189412 252431,189631  259122)',-1),'Graeme Ave');
INSERT INTO CALLES_GEOM(ID,GEOM,NAME) VALUES (1, GeometryFromText
('LINESTRING(190131  224148,190871  228134)',-1),'Phil Tce');
INSERT INTO CALLES_GEOM(ID,GEOM,NAME) VALUES (1, GeometryFromText
('LINESTRING(198231 263418,198213 268322)',-1),'Dave Cres');
END;
```

   El archivo puede cargar en la base de datos usando “psql”:

```
psql -d [base de datos] -f calles.sql
```

2. Usar el Cargador.

   El cargador de datos “**shp2pgsql**” convierte archivos de figuras ESRI a SQL para su inserción en una base de datos PostGIS/PostgreSQL. El cargador tiene varios modos de operación que se seleccionan con los parámetros desde linea de

comando:

-d Elimina la tabla de la base de datos antes de crear la tabla con los datos del archivo de figuras.

-a Añade los datos del archivo de figuras a las tabla de la base de datos. El fichero debe tener los mismos atributo que la tabla.

-c Crea una nueva tabla y llena esta con el archivo de figuras. Este es el modo por defecto.

-D Crea una tabla nueva llenándola con los datos del fichero de formas. Pero usa el formato *dump* para la salida de datos que es mas rápido que el *insert* de SQL. **Se usará este para conjuntos de datos largos.**

-s<SRID> Crea y rellena una tabla geométrica con el SRID que se le pasa como parámetro.

Ejemplos:

```
shp2pgsql shapecalles tablacalles callesdb>calles.sql
psql -d  callesdb -f calles.sql
```

Puedo hacerlo de una vez usando un pipe:

```
shp2pgsql shapecalles tablacalles callesdb | psql -d callesdb
```

4. Recuperar datos GIS.

Al igual que para la inserción de datos también tenemos 2 formas para recuperar datos. Mediante SQL o con un cargador de archivos de figuras.

1. Usar SQL.

   La forma mas directa de hacerlo es usando un SELECT. Ver Ilustración 2.

```
db=# SELECT id, AsText(geom) AS geom, name FROM ROADS_GEOM;
 id | geom                                          | 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)
```

Ilustración 2.Consulta usando un select sobre una base de datos postGIS

Algunas veces es necesario recortar el numero de registros devueltos. En el caso de restricciones basadas en los atributos usamos la misma sintaxis que para tablas

no espaciales. Para restricciones espaciales tenemos los siguientes operadores:

&& Indica cuando la caja que contiene una geometría se superpone a la caja de otra.

~= Me dice si dos geometrías son idénticas. Como 'POLIGON((0 0,1 1,1 0,0 0))'~='POLIGON((0 0,1 1,1 0,0 0))'

= Indica si las cajas circunscritas de dos geometrías son iguales.

Para usar estos operadores debemos cambiar la representación del formato de texto a geometrías usando la función **GeometryFromText()**.Ejemplo:

```
SELECT ID,NAME
 FROM CALLES_GEOM
WHERE
GEOM~=GeometryFromText('LINESTRING(191232  243118,191108 243242)',-1);
```

Retorna los registros cuya geometría es igual a la dada.

Cuando usamos el operador **&&**, podemos especificar como característica de comparación una BOX3D o una GEOMETRY.

```
SELECT ID, NAME
FROM CALLES_GEOM
WHERE
GEOM && GeometryFromText('POLYGON((191232 243117,191232 243119,191234
243117,191232 243117))',-1);
```

La consulta mas usada es la basada en marcos, que usan los clientes para mostrar un marco del mapa. Podemos usar un BOX3D para el marco:

```
SELECT AsText(GEOM) AS GEOM
FROM ROADS_GEOM
WHERE
GEOM && GeometryFromText('BOX3D(191232 243117,191232 243119)'::box3d,-1);
```

El -1 indica que no se especifica ningún SRID.

2. Usar un cargador(DUMPER).

Pgsql2shp conecta directamente con la base de datos y convierte una tabla en un archivo de figuras. La sintaxis es:

pgsql2shp [<opciones>] <basededatos> <tabla>

Opciones:

-d Escribe un archivo de figuras 3D siendo el 2D el que tiene por defecto.

-f<archivo> archivo de salida.

-p<puerto> puerto de conexión con la base de datos.

-h<host> host donde esta la base de datos.

-p<password> password para el acceso

-u<user> usuario de acceso.

-g<columna geometría> Si la tabla tiene varias columnas geométricas, selecciona la columna geométrica a usar.

El operador && solo chequea las zonas de superposición de las cajas exteriores, pero se puede usar la función *truly_inside()* para obtener solo aquellas características que pasan por la caja de búsqueda. Se pueden combinar el operador && para hacer un búsqueda indexada y *truly_inside()* para precisar el conjunto de resultados. Por ahora *truly_inside* solo funciona para buscar en cajas, no en cualquier tipo de geometría.

SELECT [COLUMN1],[COLUMN2], AsText([GEOMETRYCOLUMN]) FROM [TALBE] WHERE [GEOM_COLUMN]&&[BOX3D] AND truly_inside([GEOM_COLUMN],[BOX3D]);

¿ Encontrar todos los objetos que están a un radio dado de otro objeto?

La forma mas eficiente de hacer esta consulta es combinando la consulta de radio con la consulta de caja circunscrita. La consulta de caja circunscrita nos permite la consulta con índices espaciales, a la que después se le aplica la consulta de radio.

Ejemplo: Objetos que están a 100 metros del punto (1000 1000):

```
SELECT *
FROM GEOTABLE
WHERE
        GEOM && GeometryFromText('BOX3D(900 900,1100 1100)',-1)
AND
Distance(GeometryFromText('POINT(1000 1000)',-1),GEOM)<100;
```

¿ Como hacer una reproyección de coordenadas en una consulta?

Los sistemas de coordenadas fuente y destino han de estar en la tabla **SPATIAL_REF_SYS**,y el objeto que se va ha proyectar ha de tener SRID uno de los dos.

SELECT Transform(GEOM,4296) FROM GEOTABLE;

5. Construir índices.

Sin los índices cualquier búsqueda en una base de datos de gran tamaño seria interminable por tener que hacerse de forma secuencial. PostgreGIS soporta 3 tipos de índices por defecto:B-Tree,R-Tree y GIST.

- **B-Tree** se usan sobre datos que pueden ser ordenados sobre un eje;como por ejemplo, números, letras, fechas. Pero los datos GIS no pueden ordenarse sobre un eje.
- **R-Trees** divide los datos en rectángulos, subrectángulos, subsubrectángulos, etc. La implementación de PostgreSQL de los R-Tree no es tan robusta como la implementación GIST.
- **GIST** (Generalized search trees) Estos índices dividen los datos en "**cosas que están a un lado**","**cosas que se superponen**" y "**cosas que están dentro**".Además soporta un amplio rango de tipos de datos,incluidos los datos GIS. PostGIS usa índices R-Tree sobre GIST para indexar los datos GIS.

5.1. Índices GIST.

GIST proporciona un Árbol de busca generalizado y una forma generalizada de indexación. Además de proporcionar indexación sobre datos GIS, GIST aumenta la velocidad de las búsquedas con toda clase de estructuras irregulares(datos espectrales,arrays de enteros,etc) las cuales no se pueden tratar con la indexación basada en B-Tree.

Cuando la tabla espacial excede unos pocos cientos de filas, y si las búsquedas no solo están basadas en atributos, es recomendable crear un índice para incrementar la velocidad de las búsquedas.

La sintaxis par construir un índice sobre una columna geométrica es:

```
CREATE INDEX [nombredelindice] ON [nombretabla]
USING  GIST ([campogeometrico] GIST_GEOMETRY_OPS);
```

La creación de un índice es una tarea intensiva computacionalmente hablando:una tabla de 1 millón de registros en una maquina a 300MHZ,requiere 1 hora para construir un índice GIST. Por esto es importante que después de construir un índice forzar a PostgreSQL a guardar las estadísticas de las tabla, que después serian usadas para optimizar los planes de consulta:

```
VACUUM ANALYZE;
```

Los Índices GIST tienen dos ventajas sobre los R-Tree implementados en PostgreSQL:

- GIST pueden indexar columnas con valores nulos.
- Soportan el concepto de *lossiness*(permite almacenar la parte importante del objeto grande en un índice) el cual es importante cuando tratamos con objetos de mas de 8K.

5.2. Usar índices.

Una vez construido el índice el planificador de consultas decide cuando usar el índice de forma transparente. Pero para los Índices GIST el planificador de consultas de PosgreSQL no esta optimizado y realiza búsquedas secuenciales aun teniendo el índice.

Si no se están usando los Índices GIST en la base de datos podemos hacer 2 cosas:

1. Asegurarnos de que hemos ejecutado "**VACUUM ANALYZE [tabla]"** en las tablas que nos dan problemas. *Vacuum analyze* recoge estadísticas sobre el numero y distribución de los valores en la tabla, esto ayuda al planificador de consultas a tomar la decisión de usar el índice. Es una buena costumbre ejecutar *Vacuum analyze* de forma regular.
2. Si el uso de vacuum no funciona debemos forzar al planificador a usar la información del índice con el comando "**SET=OFF**". Este comando ha de usarse con moderación y solo con consultas espaciales . Una vez terminada la consulta tenemos que volver a poner a on "**ENABLE_SEQSCAN**", para que se use de forma normal en otras consultas. En la versión 0.6 de postGIS no es necesario forzar al planificador a usar el índice usando **ENABLE_SEQSCAN.**

6. Clientes Java(JDBC).

Los clientes java pueden acceder a los objetos geométricos de PosTGIS directamente como una representación textual o mediante JDBC con los objetos contenidos en postGIS. Para usar el jdbc debemos tener en el CLASSPATH el paquete del driver JDC "**postgresql.jar**".

Ejemplo:

```
 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
    {
      /*
```

```
        * Cargo el driver JDBC y establezco la conexion.
        */
        Class.forName("org.postgresql.Driver");
        String url = "jdbc:postgresql://localhost:5432/database";
        conn = DriverManager.getConnection(url, "postgres", "");

        /*
        * Añado los tipos geometricos a la conexion.Debo
        * hacer un castin  a el pgsql-specific conexion implementacion antes de
                llamar al metodo addDataType()
        */

((org.postgresql.Connection)conn).addDataType("geometry","org.postgis.PGgeometry");

((org.postgresql.Connection)conn).addDataType("box3d","org.postgis.PGbox3d");

         /*
        * creo una consulta y la ejecuto
        */
        Statement s = conn.createStatement();
        ResultSet r = s.executeQuery("select AsText(geom) as geom,id from geomtable");
        while( r.next() )
        {
          /*
           * recupera el objeto como una geometría y hace un casting a un tipo
          *geométrico.
          */
          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();
```

```
    }
  }
}
```

El objeto *Pggeometry* es un objeto envoltorio que contiene objetos topológicos específicos (subclases de la clase abstracta *Geometry*) dependiendo del tipo: *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());
    }
  }
}
```

7. Referencia PostGIS.

Funciones OpenGIS:

1. **AddGeometryColumn(varchar,varchar,varchar,integer,varchar,integer)**

   Sintaxis:
   AddGeometryColumn(<nombre_db>,<nombre_tabla>,<nombre_columna>,<srid>,<type>,<dimension>)

   Añade una columna geométrica a una tabla existente. SRID debe ser un valor entero que referencia una valor en la tabla *SPATIAL_REF_SYS*. El tipo debe ser una cadena en mayúsculas que indica el tipo de geometría, como, '*POLYGON*' o '*MULTILINESTRING*'.

2. **DropGeometryColumn(varchar,varchar,varchar)**

   Sintaxis:DropGeometryColumn(<nombre_db>,<nombre_Tabla>,<nombre_columna>)

   Elimina una columna geométrica de una tabla espacial.

3. **AsBinary(geometry)**

   Devuelve la geometría pasándola a formato *well-known-binary* de OGC, usando la codificación *endian* del servidor donde se ejecuta la base de datos.

4. **Dimension(geometry)**

   Devuelve 2 si la geometría es de 2D y 3 si es 3D.

5. **Envelope(geometry)**

   Retorna un *POLYGON* que representa la caja circunscrita de la geometría.

6. **GeometryType(geometry)**

   Retorna el tipo de geometría como una cadena. Ejemplo: '*LINESTRING*','*POLYGON*','*MULTIPOINT*',etc.

7. **X(geometry)**

   Encuentra y devuelve la coordenada X del primer punto de *geometry*. Devuelve NULL si no hay puntos.

8. **Y(geometry)**

   Encuentra y devuelve la coordenada Y del primer punto de *geometry*. Devuelve NULL si no hay puntos.

9. **Z(geometry)**

   Encuentra y devuelve la coordenada Z del primer punto de geometry. Devuelve NULL si no hay puntos.

10. **NumPoints(geometry)**

Busca y devuelve el numero de puntos del primer *linestring* en la *geometry*. Devuelve NULL sino hay *linestring*.

11.PointN(geometry,integer)

Devuelve el punto enésimo en el primer *linestring* de *geometry*. Y NULL sino hay ningún *linestring.*

12.EsxteriorRing(geometry)

Devuelve el circulo exterior del primer *polygon* en la *geometry*. Y nulo sino hay polígonos.

13.NumInteriorRings(geometry)

Devuelve el numero de círculos interiores del primer polígono de la geometría. Y NULL sino hay ningún polígono.

14.InteriorRingN(geometry,integer)

Devuelve el enésimo circulo interior del primer polígono en la *geometry*. Y NULL sino hay polígonos.

15.IsClosed(geometry)

Devuelve cierto si punto final = punto inicial de la geometría.

16.NumGeometries(geometry)

Si *geometry* es una *GEOMETRYCOLLECTION* devuelve el numero de geometrías que la componen. En caso contrario devuelve NULL.

17.Distance(geometry,geometry)

Devuelve la distancia cartesiana entre dos geometrías en unidades proyectadas.

18.AsText(geometry)

Devuelve una representación textual de una geometría. Ejemplo:

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

19.SRID(geometry)

Devuelve un numero entero que es el identificador del sistema de referencia espacial de una geometría.

20.GeometryFromText(varchar,integer)

Sintaxis: GeometryFromText(<geom>,<SRID>) convierte un objeto de la representación textual a un objeto geometría.

21.GeomFromText(varchar,integer)

Igual que *GeometryFromText* .

22.SetSRID(geometry)

Establece el valor del *SRID* de una geometría al entero dado. Usado en la construcción de cajas circunscritas para consultas.

23.EndPoint(geometry)

Devuelve un objeto punto que representa el ultimo punto en la geometría.

24.StartPoint(geometry)

Devuelve un objeto punto que representa el primer punto en la geometría.

24.Centroid(geometry)

Devuelve un punto que representa el *centroide* de la geometría.

Otras funciones:

1. **A<&B**

   Devuelve verdadero si la caja que circunscribe a A superpone o esta a la derecha de la de B.

2. **A&>B**

   Devuelve verdadero si la caja que circunscribe a A superpone o esta a la izquierda de la de B.

3. **A<<B**

   Devuelve verdadero si la caja que circunscribe a A esta estrictamente a la derecha de la de B.

4. **A>>B**

   Devuelve verdadero si la caja que circunscribe a A esta estrictamente a la izquierda de la de B.

5. **A~=B**

   Es equivalente al operador “igual que”. Compara las 2 geometrías característica por característica y si todas coinciden devuelve verdadero.

6. **A~B**

   Devuelve verdadero si la caja circunscrita de A esta contendida en la de B.

7. **A&&B**

   Si la caja circunscrita de A se superpone a la de B devuelve Verdadero.

8. **area2d(geometry)**

   Devuelve el área de una geometría si es un *POLYGON* o *MULTIPOLYGON*.

9. **asbinary(geometry,'NDR')**

   Devuelve la geometría en el formato binario de OGC, usando codificacion little-endian. Se usa para sacar datos de la bd sin convertirlos a texto.

10.asbinary(geometry,'XDR')

Devuelve la geometría en el formato binario de *OGC*, usando codificación *big-endian*. Se usa para sacar información de la base datos sin convertirla a texto.

11.box3d(geometry)

Devuelve una *BOX3D* que representa la máxima extensión de la geometría.

12.collect(geometry)

Devuelve un objeto *GEOMETRYCOLLECTION* a partir de un conjunto de geometrías. Es una función de Agregación en la terminología de PostgreSQL.

Ejemplo:

```
SELECT COLLECT(GEOM) FROM GEOTABLE GROUP BY ATTRCOLUMN
```

Devuelve una colección separada para cada valor distinto de *ARRTCOLUMN*.

13.distance_spheroid(point,point,spheroid)

Devuelve la distancia lineal entre 2 latitud/longitud puntos dados de un spheroid. Ver el apartado length_spheroid().

14.extent(geometry)

Es una función de Agregación en la terminología de PostgreSQL. Trabaja sobre una lista de datos, de la misma manera que las funciones sum() y mean().

Ejemplo:

```
SELECT EXTENT(GEOM) FROM GEOMTABLE
```

Devuelve una BOX3D dando la máxima extensión a todas las características de la tabla.

```
SELECT EXTENT(GEOM) FROM GEOMTABLE GROUP BY CATEGORY
```

Devuelve un resultado extendido para cada categoria.

15.find_srid(varchar,varchar,varchar)

Sintaxis: find_srid(<db/esquema>,<tabla>,<columna>)

Devuelve el *SRID* de una columna dada buscando esta en la tabla *GEOMETRY_COLUMNS*. Si la columna geométrica no ha sido añadida con la función AddGeometryColumns() no funcionará.

16.force_collection(geometry)

Convierte una geometría en una *GEOMETRYCOLLECTION*. Esto se hace para simplificar la representación WKB .

17.force_2d(geometry)

Fuerza la geometría a 2D así la representación de salida tendrá solo las coordenadas X e Y. Se usa porque OGC solo especifica geometrías 2D.

18.force_3d(geometry)

Fuerza la geometría a 3D. Así la todas las representaciones tendrán 3 coordenadas X,Y y Z.

19.length2d(geometry)

Devuelve la longitud 3d de la geometría si es una *linestring* o *multilinestring*.

20.length3d(geometry)

Devuelve la longitud 2d de la geometría si es una *linestring* o *multilinestring*.

21.length_spheroid(geometry,spheroid)

Calcula la longitud de una geometría en un elipsoide. Se usa si las coordenadas de la geometría están en latitud/longitud. El elipsoide es un tipo separado de la base de datos y se puede construir como:

SPHEROID [<NAME>,<SEMI-MAJOR AXIS>,<INVERSE FLTTENING>]

Ejemplo:

```
SPHEROID [“GRS_1980”,6378137,298,257222101]
```

Ejemplo de calculo:

```
SELECT
LENGTH_SPHEROID(
    geometry_column,
    'SPHEROID[“GRS_1980”,6378137,298.257222101]'
)
FROM geometry_table;
```

22.length3d_spheroid(geometry,spheroid)

Calcula la longitud de una geometría en un elipsoide, teniendo en cuenta la elevación.

23.max_distance(linestring,linestring)

Devuelve la distancia mas larga entre dos *linestring*.

24. mem_size(geometry)

Retorna el tamaño en bytes de la geometría.

25.npoints(geometry)

Devuelve el numero de puntos en la geometría.

26.nrings(geometry)

Si la geometría es un polígono o multipolígono devuelve el numero de círculos de la geometría.

27.num_sub_objects(geometry)

Devuelve el numero de objetos almacenados en la geometría. Se usa en Multigeometrías y *GEOMETRYCOLLECTIONs*.

28.perimeter2d(geometry)

Devuelve el perímetro 2d de la geometría, si esa geometría es un polígono o un multipolígono.

29.perimeter3d(geometry)

Devuelve el perímetro 3d de la geometría, si esa geometría es un polígono o un multipolígono.

30.point_inside_circle(geometry,float,float,float)

Sintaxis:point_inside_circle(<geometry>,<circle_center_x>,<circle_center_y>,<radius>)

Devuelve verdadero si la geometría es un punto y esta dentro del circulo.

31.postgis_version()

Devuelve la versión de las funciones postgis instaladas en la bases de datos.

32.summary(geometry)

Devuelve un resumen en texto del contenido de esa geometría.

33.transform(geometry,integer)

Devuelve una nueva geometría con sus coordenadas transformadas la SRID dada por el parámetro integer. SRID debe existir en la tabla SPATIAL_REF_SYS.

34.translate(geometry,float8,float8,float8)

Traslada la geometría a la nueva localización usando los valores pasados como desplazamientos X,Y,Z.

35.truly_inside(geometryA,geometryB)

Devuelve verdadero si alguna parte de B esta dentro de la caja circunscrita de A.

36.xmin(box3d) ymin(box3d) xmin(box3d)

Devuelve la mínima coordenada de la caja circunscrita.

37.xmax(box3d) ymax(box3d) zmax(box3d)

Devuelve la máxima coordenada de la caja circunscrita.