

PostGIS

Bases de datos espaciales

José Carlos Martínez Llario

PostGIS

Bases de datos espaciales

© 2012-2025 José Carlos Martínez Llario.

Todos los nombres propios de programas, sistemas operativos, empresas, etc. que aparecen en este libro son marcas registradas de sus respectivas compañías u organizaciones.

Todos los derechos reservados. Queda prohibida la reproducción, distribución, comercialización, transformación, y en general, cualquier otra forma de explotación, por cualquier procedimiento, de todo o parte de los contenidos de esta obra sin autorización expresa y por escrito de sus autores.

Revisión 5. <https://cartosig.webs.upv.es>

Prólogo

Versiones de los programas analizados

En esta revisión del libro se ha adaptado el contenido y comprobado todos los ejemplos a la versión 17 de PostgreSQL y la 3.5 de PostGIS. Si el lector dispone de versiones más actuales, puede utilizarlas sin problema, ya que las funcionalidades de ambos productos son muy estables a lo largo del tiempo, y en el caso de alguna discrepancia con la publicación, los cambios son publicados en el blog del autor <https://cartosig.webs.upv.es>.

A quién va dirigido este libro

Este libro está diseñado para cubrir diferentes perfiles, desde un usuario que desea introducirse en el mundo de las bases de datos espaciales y PostGIS, hasta un perfil más experto que, aunque posee ya conocimientos de PostGIS necesita realizar análisis espaciales avanzados o incluso extender PostGIS con nuevas funcionalidades.

Está enfocado como un **manual de aprendizaje**, guiado con una gran cantidad de ejemplos prácticos con cartografía real, con el objetivo de que un usuario tradicional de SIG de escritorio pueda migrar a PostGIS de una forma no traumática.

Desde el ámbito de la producción cartográfica se describen los problemas de la precisión cartográfica de los datos y el análisis espacial, las validaciones topológicas, reglas de topología, y otros aspectos que posibilitan la utilización de PostGIS no solo como un gestor de bases de datos espacial sino también como una herramienta de producción cartográfica.

Se ha tratado de realizar una publicación que sea útil tanto para un usuario experto en ciencias de la computación y bases de datos, pero que desconoce cómo gestionar de forma apropiada la cartografía, como para un usuario experto en ciencias cartográficas pero que, al ser reticente a dejar los SIG de escritorio, está sacrificando el conocimiento de unas funcionalidades espaciales excepcionales.

Conocimientos previos

Es aconsejable disponer de unos conocimientos previos a nivel de usuario de Sistemas de Información Geográfica, aunque no son necesarios para la comprensión del texto, resultan recomendables para la lectura fluida del mismo.

Si el lector no posee conocimientos sobre el lenguaje SQL o estos son muy superficiales, es necesario que, después de leer el capítulo A introductorio y antes de pasar al capítulo B, estudie con detalle el apartado I 2 sobre SQL.

Estructura

PostGIS: Bases de datos espaciales se compone de nueve capítulos. Se sugiere que el lector estudie los capítulos B y C en profundidad, y que sea más selectivo en el capítulo D según sus necesidades. El capítulo E, dedicado a la programación, no es imprescindible; aunque, su estudio es recomendable debido a las oportunidades que ofrece. El capítulo F presenta propuestas útiles para mejorar el aprendizaje en PostGIS. Los capítulos G y H abordan análisis *raster*, topología persistente y *pgrouting*, y pueden consultarse según el interés del usuario. Finalmente, el capítulo I incluye una guía de SQL, notas de administración y soluciones a problemas propuestos.

- *A Introducción*

Se inicia la publicación con este breve capítulo, donde se estudia la normativa relacionada con PostGIS y el modelo de geometrías seguido. Se aborda la instalación y se introducen los clientes SQL *pgadmin* y *psql*.

- *B Núcleo*

En este capítulo se describe la parte fundamental de la funcionalidad PostGIS y del modelo de geometría utilizado siguiendo los estándares del OGC y normas ISO. Se estudian las relaciones espaciales con detalle, diferentes programas de utilidad para importar, exportar datos PostGIS, indexación espacial, tablas y vistas espaciales, etc.

- *C Análisis espacial*

Esta sección del libro tiene como objetivo capacitar al lector en la realización de análisis espaciales con PostGIS. Después de estudiar los operadores espaciales, se combinan para realizar operaciones comunes de análisis vectorial en SIG de escritorio, como superposición de capas, extracción, proximidad y disolución.

Este aprendizaje es fundamental para que el lector comprenda la potencia de PostGIS y su flexibilidad en comparación con otros SIG de escritorio. Aunque el contenido puede resultar complejo para usuarios no expertos en ciencias de la computación y bases de datos, la explicación detallada permitirá a los lectores realizar consultas espaciales complejas al finalizar el capítulo

- *D Validación cartográfica*

Este capítulo aborda conceptos avanzados en la gestión cartográfica de datos, enfatizando la importancia de la precisión cartográfica, el uso de tolerancias, así como la validación y corrección de geometrías, aspectos frecuentemente ignorados por usuarios sin experiencia.

Se destaca el diseño y la aplicación de reglas de topología entre geometrías en una o varias capas, con el fin de identificar y corregir errores cartográficos.

Además, se examina la optimización del análisis espacial mediante un diseño adecuado, el uso de técnicas de rejilla, el análisis espacial sobre el esferoide y los cambios de sistema de referencia que involucran diferentes datum.

- *E Programación*

Aunque no es necesario saber programar en PostGIS, muchos usuarios pueden sentirse interesados en adquirir esta habilidad. En esta sección se explica cómo crear funciones personalizadas para ampliar la funcionalidad espacial de PostGIS.

Los usuarios descubrirán que es sencillo aprender a desarrollar procedimientos almacenados con *PL/PgSQL*, lo que abre un mundo de posibilidades, especialmente en la gestión de modelos de datos. Al implementar disparadores (*triggers*) junto con funciones *PL/PgSQL* personalizadas, es posible dotar a los modelos de datos cartográficos de un comportamiento dinámico e incorporar lógica en la base de datos.

- *F Miscelánea espacial*

Este apartado aglutina parte del resto de la funcionalidad de PostGIS no vista en los capítulos anteriores y que generalmente se referencia en otros capítulos previos como el análisis espacial 3D y las geometrías curvas.

Es especialmente interesante el uso de funciones ventana, que, junto con las consultas laterales, facilitan mucho cierto tipo de consultas espaciales. Otros temas estudiados son la dependencia funcional, importación de datos OSM, controles de versiones, etc.

El capítulo se completa con las particiones declarativas y las consultas paralelas, las cuales se pueden utilizar para incrementar el rendimiento de la base de datos espacial.

- *G Análisis raster*

Se estudia en profundidad la extensión de PostGIS para la gestión y análisis de datos en formato *raster*. Se repasan las operaciones de análisis típicas raster (reclasificaciones, mapas derivados de modelos digitales de elevación, álgebra de mapas, etc.) y mixtas vectorial-raster.

- *H Extensiones*

Este capítulo aborda dos extensiones de PostGIS: Topología persistente (*topology*) y análisis de redes (*pgrouting*). La extensión *topology* gestiona relaciones geométricas complejas, garantizando la integridad de los datos y facilitando el diseño de modelos cartográficos. Por su parte, *pgrouting* ofrece funcionalidades avanzadas para calcular rutas óptimas y realizar análisis de redes.

- *I Anexos*

Los anexos están divididos en tres bloques: Un primer bloque sobre administración de PostgreSQL donde se introducen temas como la utilización de variables del sistema, métodos de autenticación, localización, gestión de usuarios, etc.

Un segundo bloque sobre el lenguaje SQL, donde en apenas 40 páginas se hace un recorrido imprescindible para los usuarios que no conozcan este lenguaje y que es necesario estudiar con detalle antes de adentrarse en el mundo PostGIS. Un tercer bloque con las soluciones de todos los problemas propuestos a lo largo de la publicación cierra el capítulo.

Material complementario

Desde el blog oficial del autor¹, se pueden **descargar los datos necesarios** (cartografía, código SQL, etc.) para la realización de todos los ejercicios del libro.

Uso del libro

Se utiliza tipografía cursiva en los siguientes casos:

- Nombres de tablas, columnas y métodos o funciones de PostGIS.
- Palabras en inglés generalmente provenientes del ámbito informático, como *prompt*, *bugs*, *raster*, *online*, etc.
- Palabras no aceptadas por la Real Academia de la Lengua, pero usadas comúnmente en el ámbito de la geometría computacional o informático: *renderizado*, *modificación*, *cardinalidad*, etc. Aunque estas palabras no están admitidas, se ha decidido mantenerlas por su carácter aclaratorio.

A lo largo de la publicación se insta al lector a ejecutar comandos o sentencias SQL:

- Los ejemplos de sentencias SQL se encabezan por el nombre de la base de datos seguido del símbolo '#', como aparece en el *prompt* del cliente *psql* de PostgreSQL:

```
s1=# create table viarial (gid serial primary key, tipo integer,
                             geom geometry (multipolygon, 25830) );
```

- Los ejemplos de comandos PostgreSQL, PostGIS o cualquier otra utilidad lanzados desde una terminal o símbolo del sistema se encabezan con el texto: '*consola*>':

```
consola> psql -U postgres -f ruta_a_postgis.sql -d s0
```

Existen dos tipos de ejercicios que se introducen en el texto con los siguientes iconos:



Ejercicios planteados y resueltos en la misma lección.



Problemas propuestos, con la solución al final del texto.

¹ <https://cartosig.webs.upv.es>

Acerca del autor

El Dr. José Carlos Martínez Llario es profesor en la Universidad Politécnica de Valencia desde el año 2000, donde imparte asignaturas especializadas en sistemas de información geográfica, producción cartográfica y bases de datos espaciales. En 2003, obtuvo su doctorado en ingeniería en geodesia y cartografía en la misma institución. Sus principales líneas de investigación en los últimos años se han centrado en bases de datos espaciales, modelos de datos cartográficos e infraestructuras de datos espaciales.

Entre sus trabajos recientes destaca el diseño y desarrollo de Jaspa, un software de código abierto que actúa como extensión espacial para bases de datos relacionales, ofreciendo funcionalidades similares a las de PostGIS y un sistema de reglas de topología. En el ámbito académico, Martínez Llario ha mantenido una actividad investigadora continua, destacándose como investigador principal en proyectos fundamentales del Ministerio de Economía y Competitividad de España, y ha establecido convenios de I+D con empresas e instituciones públicas.

Ficha pública del autor en <https://www.upv.es/ficha-personal/jomarlla>

Quiero expresar mi agradecimiento a las comunidades de usuarios y a las instituciones que apuestan por el software libre, ya que esta publicación no habría sido posible sin su esfuerzo. En especial, a los miembros y colaboradores del equipo de PostGIS que de forma desinteresada han resuelto todas mis dudas siempre de manera amable y eficaz.

Cursos online de PostGIS certificados por la UPV

El autor de esta publicación junto con otros miembros del grupo CartoSIG UPV, imparte varios cursos online de PostGIS, Infraestructuras de Datos Espaciales (IDE) y Sistemas de Información Geográfica (SIG), cursos certificados por el Centro de Formación Permanente de la Universitat Politècnica de València.

Se trata de unos cursos completos, de calidad ampliamente contrastada y con cien horas reales de aprendizaje, que se pueden realizar completamente a distancia. Su contenido está basado en la experiencia de docencia presencial oficial de universidad e investigación del autor de esta publicación.

Son cursos que han realizado miembros de las principales empresas u organizaciones públicas relacionadas con la cartografía a nivel nacional y que gozan de una buena reputación.

Esta publicación constituye un material de apoyo o consulta de PostGIS. Para un aprendizaje completo desde el inicio, práctico, con casos reales, consolidando conocimientos y siguiendo una metodología docente ya probada, se recomienda al lector realizar el curso online de PostGIS.

Más información en: <https://cartosig.webs.upv.es/cursos>

Índice breve de contenido

A	Introducción.....	21
1.	PRIMEROS PASOS.....	23
2.	CARTOGRAFÍA Y CÓDIGO FUENTE DE LOS EJEMPLOS.....	36
B	Núcleo.....	37
1.	LA BASE DE DATOS ESPACIAL.....	39
2.	TIPOS DE GEOMETRÍA.....	51
3.	EJEMPLOS DEL CAPÍTULO.....	69
4.	MODELO <i>SIMPLE FEATURES</i> O FENÓMENOS SIMPLES.....	75
5.	RELACIONES ESPACIALES.....	90
6.	INDEXACIÓN ESPACIAL.....	104
7.	CREACIÓN DE TABLAS Y VISTAS PARA ALMACENAR CONSULTAS ESPACIALES.....	115
C	Análisis espacial.....	125
1.	INTRODUCCIÓN.....	127
2.	OPERADORES ESPACIALES.....	128
3.	OPERACIONES DE SUPERPOSICIÓN (<i>OVERLAY</i>).....	137
4.	OPERACIONES DE EXTRACCIÓN.....	145
5.	PROXIMIDAD.....	147
6.	GENERALES.....	163
7.	GENERALIZACIÓN.....	170
8.	TRANSFORMACIÓN Y EDICIÓN DE COORDENADAS.....	177
9.	CONVERSIONES.....	183
10.	REFERENCIA LINEAL (LRS).....	195
D	Validación cartográfica.....	199
1.	OPTIMIZACIÓN DEL ANÁLISIS ESPACIAL.....	201
2.	PROYECCIONES Y TRANSFORMACIONES ENTRE <i>DATUM</i>	214
3.	TOLERANCIA EN EL ANÁLISIS ESPACIAL.....	220
4.	VALIDACIÓN DE LAS GEOMETRÍAS.....	228
5.	ANÁLISIS ESPACIAL SOBRE EL ESFEROIDE.....	233
6.	VALIDACIÓN CARTOGRAFICA CON REGLAS DE TOPOLOGÍA.....	243
E	Programación.....	267
1.	INTRODUCCIÓN.....	269
2.	SCRIPTS <i>PL/PGSQL</i> EN POSTGIS.....	272

F	Miscelánea espacial.....	319
1.	ARRAYS, AGREGADOS Y SET RETURNING DE GEOMETRÍAS.....	321
2.	TIPOS COMPUESTOS.....	324
3.	COMPORTAMIENTO MULTI-GEOMETRÍAS.....	326
4.	DEPENDENCIA FUNCIONAL (GROUP BY).....	328
5.	FUNCIONES DE VENTANA ESPACIALES.....	330
6.	COPIA DE SEGURIDAD DE LA BASE DE DATOS ESPACIAL.....	335
7.	OPERACIONES 3D.....	340
8.	GEOMETRÍAS CURVAS.....	349
9.	TIPOS DE DATOS DE CAJAS.....	353
10.	IMPORTACIÓN DE DATOS OSM.....	355
11.	CONTROL DE VERSIONES EN POSTGIS.....	362
12.	PARTICIONES DE DATOS.....	369
13.	CONSULTAS ESPACIALES PARALELAS.....	378
G	Análisis raster.....	385
1.	INTRODUCCIÓN.....	387
2.	TIPO RASTER.....	388
3.	CAPAS RASTER.....	404
4.	ANÁLISIS DE CAPAS TESELADAS.....	416
H	Extensiones.....	449
1.	PGROUTING.....	451
2.	TOPOLOGÍA PERSISTENTE.....	470
I	Anexos.....	499
1.	NOTAS SOBRE ADMINISTRACIÓN.....	501
2.	SQL.....	531
3.	SOLUCIONES.....	570
J	Recursos y bibliografía.....	599
1.	RECURSOS PRÁCTICOS Y TUTORIALES.....	601
2.	BIBLIOGRAFÍA.....	602

Índice de contenido

A	Introducción.....	21
1.	PRIMEROS PASOS.....	23
1.1.	<i>Normativa relacionada</i>	24
1.2.	<i>Instalación</i>	25
	PostgreSQL.....	25
	Instalación de PostGIS.....	28
1.3.	<i>Cientes SQL</i>	29
	Clientes de texto: <i>psql</i>	29
	Clientes gráficos: <i>pgadmin</i>	32
1.4.	<i>Tipos básicos de datos</i>	33
1.5.	<i>Notas antes de empezar</i>	35
	Creación de una nueva base de datos.....	35
	Cambio de contraseña.....	35
	Conocimientos previos.....	35
2.	CARTOGRAFÍA Y CÓDIGO FUENTE DE LOS EJEMPLOS.....	36
B	Núcleo.....	37
1.	LA BASE DE DATOS ESPACIAL.....	39
1.1.	<i>Creación base de datos espacial sin utilizar extensions</i>	39
	Módulo principal y vectorial.....	39
	Módulo de <i>raster</i>	41
	Módulo de topología persistente.....	41
	Otros módulos.....	41
	Nota sobre los ficheros SQL de PostGIS.....	41
1.2.	<i>Creación base de datos espacial utilizando extensions</i>	42
1.3.	<i>Comprobación de la base de datos espacial</i>	43
1.4.	<i>Funciones extra y plantillas</i>	44
	Creación de una plantilla espacial.....	44
1.5.	<i>Metadatos sobre los CRS</i>	45
1.6.	<i>Creación y borrado de una tabla espacial</i>	46
	Utilizando <i>typmod</i>	46
	Utilizando restricciones de tipo <i>check</i>	49
1.7.	<i>Metadatos de las columnas de geometría</i>	50
2.	TIPOS DE GEOMETRÍA.....	51
	Vértices con Z, M o ZM.....	52
2.1.	<i>Creación e inserción de geometrías</i>	52
	Conversiones a otros formatos.....	55
	Conversiones automáticas al tipo <i>geometry</i>	56
	Relajación de las restricciones de una columna de geometría.....	56

2.2.	<i>Importación y exportación de cartografía</i>	57
	Comandos <i>shp2pgsql</i> , <i>pgsql2shp</i> y <i>shp2pgsql-gui</i>	57
	Trabajo con esquemas: <i>search_path</i>	61
	GDAL/OGR (<i>ogr2ogr</i> y <i>ogrinfo</i>)	62
	Problemas de conversión entre juegos de caracteres	66
	Importación de datos OSM a PostGIS	67
	Utilización de SIG de escritorio	67
3.	EJEMPLOS DEL CAPÍTULO	69
3.1.	<i>Datos cartográficos utilizados en los ejemplos</i>	69
3.2.	<i>Visualización y edición gráfica de capas PostGIS</i>	69
	Conclusiones	71
4.	MODELO SIMPLE FEATURES O FENÓMENOS SIMPLES	75
4.1.	<i>Esquema de herencia de las geometrías</i>	75
	<i>JTS Builder</i>	76
4.2.	<i>Dimensión de una geometría</i>	78
4.3.	<i>Interior, contorno y exterior de las geometrías</i>	78
	Contorno de una <i>MultiCurve</i>	80
4.4.	<i>Definición de las geometrías básicas</i>	80
	Subconsultas en PostGIS (<i>Subselect</i> y <i>CTE</i>)	81
	<i>ST_Point</i> y <i>ST_Multipoint</i>	83
	<i>ST_Curve</i> <- <i>ST_Linestring</i>	83
	<i>ST_MultiCurve</i> <- <i>ST_MultiLinestring</i>	84
	<i>ST_Surface</i> <- <i>ST_CurvePolygon</i> <- <i>ST_Polygon</i>	85
	<i>ST_MultiSurface</i> : <i>ST_MultiPolygon</i>	88
5.	RELACIONES ESPACIALES	90
5.1.	<i>Matriz DE-9IM</i>	90
	Uso de patrones DE-9IM personalizados	91
5.2.	<i>Predicados espaciales</i>	91
	<i>ST_Disjoint</i> , <i>ST_Intersects</i>	92
	Resumen de los predicados	93
	<i>ST_Touches</i>	94
	<i>ST_Crosses</i>	95
	<i>ST_Overlaps</i>	95
	<i>ST_Equals</i> versus igualdad no topológica	96
	<i>ST_Covers</i> , <i>ST_CoveredBy</i>	99
5.3.	<i>Ejemplos de predicados espaciales</i>	100
6.	INDEXACIÓN ESPACIAL	104
6.1.	<i>Creación y utilización de índices espaciales</i>	104
6.2.	<i>Otros operadores GiST sobre cajas y KNN</i>	106
	Otros operadores de comparación de cajas	106
	Operadores que utilizan KNN	106
6.3.	<i>Planificador</i>	107
6.4.	<i>Predicados espaciales con el operador && embebido</i>	109
6.5.	<i>Indexación espacial GiST 3D</i>	110
6.6.	<i>Otro tipo de índices espaciales</i>	112
	Índices BRIN	112
	Índice SP-GiST	113
6.7.	<i>Espacio ocupado por los índices</i>	114

7.	CREACIÓN DE TABLAS Y VISTAS PARA ALMACENAR CONSULTAS ESPACIALES	115
7.1.	<i>Almacenar resultados en tablas espaciales</i>	115
	Procedimiento riguroso	115
	Procedimiento práctico (CAST para los tipos de campos)	116
	Ejemplos.....	116
	Copia de la estructura de una tabla.....	118
7.2.	<i>Utilización de vistas espaciales</i>	118
	Creación de reglas para actualizar vistas	119
	Capa de eventos.....	122
	Vistas como control dinámico de la calidad cartográfica	122
C	<i>Análisis espacial</i>	125
1.	INTRODUCCIÓN	127
	Utilización de tablas en los ejemplos	127
2.	OPERADORES ESPACIALES.....	128
2.1.	<i>Ejemplos gráficos</i>	130
2.2.	<i>Tipos de geometrías devueltas</i>	132
2.3.	<i>Homogeneización de las geometrías devueltas</i>	133
	Utilización de <i>STX_Extract</i>	136
3.	OPERACIONES DE SUPERPOSICIÓN (<i>OVERLAY</i>).....	137
3.1.	<i>Intersección (Intersect)</i>	137
	Entrada: polígonos, salida: líneas	138
	Entrada: polígonos y líneas, salida: líneas	138
	Otros casos	138
3.2.	<i>Borrado (Erase)</i>	139
	Polígonos A que son borrados parcial o totalmente por polígonos B.....	140
	Polígonos A que no presentan solape con B y se conservan íntegros.....	140
	Borrado en un solo paso	141
3.3.	<i>Superposición (Overlay)</i>	141
3.4.	<i>Identidad (Identity)</i>	143
3.5.	<i>Actualización (Update)</i>	144
4.	OPERACIONES DE EXTRACCIÓN	145
4.1.	<i>Recorte (Clip)</i>	145
4.2.	<i>Selección (Select)</i>	146
5.	PROXIMIDAD.....	147
5.1.	<i>Área de influencia (Buffer)</i>	147
5.2.	<i>Selecciones según distancias</i>	149
	<i>ST_DWithin</i>	150
5.3.	<i>Tabla de proximidad (Near table)</i>	151
5.4.	<i>Vecinos más próximos a una única geometría</i>	152
5.5.	<i>Vecinos más próximos a una capa (subconsulta anidada)</i>	153
	Mediante <i>subselects</i> anidados	153
	Mediante <i>subselect</i> y agregado de mínima distancia con identificador	154
5.6.	<i>Vecinos más próximos a una capa (subconsultas correladas)</i>	155
	Obtener los n vecinos más próximos	157
5.7.	<i>Vecinos más próximos a una capa (consultas laterales)</i>	159
5.8.	<i>Vecinos más próximos a una capa (funciones ventana)</i>	160
5.9.	<i>Vecinos más próximos a una capa (operadores KNN)</i>	160

6.	GENERALES	163
6.1.	<i>Concatenación espacial (Spatial join)</i>	163
	Cardinalidad ríos (1) corrientes (1)	163
	Cardinalidad ríos (1) corrientes (0..n)	164
	Cardinalidad ríos (1) corrientes (0)	167
	Cardinalidad ríos (0) corrientes (1)	167
6.2.	<i>Adición (Append/Merge)</i>	169
7.	GENERALIZACIÓN	170
7.1.	<i>Disolución (Dissolve)</i>	170
	ST_Collect y ST_Union	172
	ST_UnaryUnion	173
7.2.	<i>Simplificación de geometrías</i>	173
8.	TRANSFORMACIÓN Y EDICIÓN DE COORDENADAS.....	177
8.1.	<i>Edición</i>	177
8.2.	<i>Transformaciones</i>	178
8.3.	<i>Proyecciones</i>	179
	Cambio de CRS de una capa	181
	Reproyección de una capa.....	182
9.	CONVERSIONES	183
9.1.	<i>Multigeometrías a geometrías simples</i>	183
	Funciones ‘set returning’	183
	ST_Dump.....	184
9.2.	<i>Conversión a segmentos lineales</i>	185
9.3.	<i>Conversión a entidades puntuales</i>	186
	Desde entidades lineales.....	186
	Desde entidades poligonales	188
9.4.	<i>Conversión a entidades lineales</i>	189
	Desde entidades poligonales	189
	Desde entidades puntuales.....	189
9.5.	<i>Conversión a entidades superficiales</i>	190
	Nodificación de geometrías lineales	191
9.6.	<i>Cambio de dimensión de las coordenadas</i>	193
10.	REFERENCIA LINEAL (LRS).....	195
	Referencia lineal: fracción de distancia como medida	195
	Referencia lineal: coordenadas M o Z como medida	196
	Pegando líneas.....	197
	Inserción de vértices en los puntos más cercanos.....	198

D Validación cartográfica..... 199

1.	OPTIMIZACIÓN DEL ANÁLISIS ESPACIAL	201
1.1.	<i>Diseño del análisis espacial</i>	201
1.2.	<i>Segmentación de capas mediante rejilla</i>	203
	Disoluciones.....	205
	Borrado.....	207
	Recorte	209
	Intersección	210
1.3.	<i>Robustez en el análisis espacial</i>	211
	JTS OverlayNG.....	211
	Errores geométricos de los operadores espaciales.....	212

2.	PROYECCIONES Y TRANSFORMACIONES ENTRE DATUM	214
2.1.	<i>Caso práctico: de ED50 a ETRS89</i>	215
	Modelo de 7 parámetros	215
	Modelo de rejilla NTV2	217
	Reiniciación de los parámetros.....	218
	Configuración localización biblioteca.....	219
3.	TOLERANCIA EN EL ANÁLISIS ESPACIAL	220
3.1.	<i>Precisión Cartográfica y la tolerancia</i>	222
3.2.	<i>Destrucción de la topología de una cartografía</i>	223
	Geometrías sin modificar en los puntos compartidos.....	225
3.3.	<i>Ajuste de vértices y segmentos entre geometrías</i>	226
4.	VALIDACIÓN DE LAS GEOMETRÍAS	228
4.1.	<i>Modelo ESRI contra modelo OGC/PostGIS</i>	229
4.2.	<i>Comandos que generan polígonos no válidos</i>	230
4.3.	<i>Corrección de geometrías no válidas</i>	231
5.	ANÁLISIS ESPACIAL SOBRE EL ESFEROIDE.....	233
5.1.	<i>Medida de distancias sobre el esferoide</i>	234
5.2.	<i>Tipo geography</i>	235
5.3.	<i>Problema directo e inverso de la Geodesia con PostGIS</i>	238
5.4.	<i>Creación de tablas espaciales</i>	239
5.5.	<i>Comparación de rendimiento geography-geometry</i>	240
5.6.	<i>Ventajas e inconvenientes de usar el tipo geography</i>	241
5.7.	<i>Análisis espacial directo sobre el esferoide</i>	242
6.	VALIDACIÓN CARTOGRÁFICA CON REGLAS DE TOPOLOGÍA	243
6.1.	<i>Validación independiente</i>	243
	Geometrías no válidas según el OGC	244
	Geometrías vacías o nulas	244
	Auto-intersecciones	244
	Sentido de los anillos.....	246
	Vértices repetidos	246
	Geometrías duplicadas	246
	Otros.....	247
6.2.	<i>Validación conjunta (una capa)</i>	248
	<i>Must not overlap</i>	249
	<i>Must not have gaps</i>	250
	<i>Must not have dangles</i>	252
	<i>Must not have dangles: distancia a la geometría más cercana</i>	253
	<i>Must not have seudos</i>	255
6.3.	<i>Validación conjunta (dos capas)</i>	256
	<i>Must contain one point</i>	257
	<i>Must be covered by layer</i>	258
	<i>Must be cross connected</i>	259
6.4.	<i>Jaspa (JAvA SPAtial)</i>	261
6.5.	<i>Definición algebraica de las reglas de topología</i>	263
6.6.	<i>Control de reglas de topología mediante disparadores</i>	266

E	Programación.....	267
1.	INTRODUCCIÓN	269
1.1.	Modelos cartográficos	271
2.	SCRIPTS PL/PgSQL EN POSTGIS	272
2.1.	Introducción al lenguaje	272
	Instalación del lenguaje.....	272
	Estructura de un método PL/PgSQL.....	272
	Declaración de variables y asignación de variables.....	274
	Condicionales.....	274
	Llamadas a otras funciones	274
	Opción <i>Strict</i>	274
	Reutilización de los resultados de una función.....	275
	Borrado de una función.....	276
	Formas alternativas de la firma de un método.....	276
	Devolviendo tipos compuestos.....	277
	<i>Arrays</i>	278
	Bucles.....	279
	Notificación de mensajes y Excepciones.....	279
	Devolviendo tablas de una columna.....	280
	Devolviendo tablas de varias columnas.....	281
	Número de argumentos variable: <i>VariaDic</i>	281
	Ejercicios espaciales complementarios	282
2.2.	Trabajando con sentencias SQL	283
	Sentencias directas	283
	Sentencias creadas de forma dinámica	288
2.3.	Funciones disparador	292
	Función disparadora en PL/PgSQL	292
	Creación del disparador SQL	293
	Resumen y principales características de los disparadores.....	295
	Ejercicios espaciales complementarios	296
2.4.	Estructura Arco/Nodo mediante disparadores.....	303
	Modelo sin crear nodos en las intersecciones.....	303
	Creación de nodos en las intersecciones.....	309
2.5.	Funciones agregadas	316
2.6.	Otras funciones.....	318
F	Miscelánea espacial.....	319
1.	ARRAYS, AGREGADOS Y SET RETURNING DE GEOMETRÍAS.....	321
1.1.	Arrays de geometrías	322
1.2.	Funciones returning set de geometrías	323
2.	TIPOS COMPUESTOS.....	324
2.1.	Tipos compuestos en PostGIS	325
3.	COMPORTAMIENTO MULTI-GEOMETRÍAS	326
4.	DEPENDENCIA FUNCIONAL (<i>GROUP BY</i>)	328
5.	FUNCIONES DE VENTANA ESPACIALES	330
5.1.	Funciones ventana de PostGIS	334

6.	COPIA DE SEGURIDAD DE LA BASE DE DATOS ESPACIAL	335
	<i>Otros métodos</i>	336
6.1.	<i>Backup y migración de una base de datos espacial</i>	337
6.2.	<i>Instalación de PostGIS en un esquema personalizado</i>	338
	Mediante extensiones: Instalación en un esquema diferente	338
	Sin el uso de extensiones: Instalación en un esquema diferente	339
7.	OPERACIONES 3D	340
7.1.	<i>Tipos de funciones espaciales 3D</i>	340
7.2.	<i>Cálculos con SFCGAL</i>	342
7.3.	<i>Nuevas geometrías superficiales</i>	345
	Ejemplos de superficies.....	346
	Visualización de geometrías superficiales	347
8.	GEOMETRÍAS CURVAS	349
8.1.	<i>Geometrías curvas de tipo multi</i>	351
8.2.	<i>Conversión entre geometrías lineales y curvas</i>	351
9.	TIPOS DE DATOS DE CAJAS.....	353
10.	IMPORTACIÓN DE DATOS OSM.....	355
10.1.	<i>Sistema de referencia usado en OSM</i>	356
10.2.	<i>Osm2Pgsql</i>	356
10.3.	<i>Osmosis</i>	360
11.	CONTROL DE VERSIONES EN POSTGIS.....	362
11.1.	<i>Ejercicio práctico de control de versiones</i>	363
	Preparación de los datos y configuración del versionado	363
	Ejercicio de edición concurrente con versionado	365
12.	PARTICIONES DE DATOS	369
12.1.	<i>Herencia de tablas</i>	369
	Limitaciones de la herencia de tablas	374
12.2.	<i>Particiones declarativas</i>	374
12.3.	<i>Tablespaces y particiones de datos</i>	377
13.	CONSULTAS ESPACIALES PARALELAS	378
13.1.	<i>Otros parámetros de configuración</i>	383

G *Análisis raster*

1.	INTRODUCCIÓN	387
2.	TIPO RASTER	388
2.1.	<i>Creación de un objeto raster vacío</i>	388
	Propiedades de un raster	389
2.2.	<i>Adición de bandas al raster</i>	391
	Propiedades de una banda	393
	Raster con varias bandas	394
	Asignación de valores a las celdas	394
2.3.	<i>Estadísticas de las bandas</i>	395
2.4.	<i>Coordenadas píxel y coordenadas terreno</i>	397
2.5.	<i>Lectura de los valores de las celdas</i>	397
2.6.	<i>Vectorización</i>	398
2.7.	<i>Rasterización de geometrías</i>	399
2.8.	<i>Exportación a otros formatos raster</i>	402

3.	CAPAS RASTER.....	404
3.1.	Importación de ficheros raster.....	404
	Visualización.....	407
3.2.	Alineamiento y teselado regular	408
	Alignment.....	408
	Regular blocking	409
3.3.	Restricciones de la capa raster.....	409
	Eliminación y creación de restricciones	411
	Restricciones de las overviews.....	412
3.4.	Vistas de metadatos.....	413
3.5.	Indexación espacial.....	414
3.6.	Exportación de capas raster con GDAL	414
4.	ANÁLISIS DE CAPAS TESELADAS	416
4.1.	Estadísticas	416
	Histogramas.....	417
4.2.	Reclasificación.....	418
	Reclasificación de múltiples bandas.....	420
4.3.	Apoyo de geometrías sobre un MDE.....	420
4.4.	Vectorización	421
4.5.	Reescalado	422
4.6.	Álgebra de mapas (una capa).....	424
	Funciones personalizadas	425
4.7.	Álgebra de mapas (dos capas).....	427
	Igual alineamiento y teselado	428
	Diferente teselado.....	431
	Unión de dos objetos raster	433
	Diferente alineamiento: remuestreo	434
4.8.	Funciones de vecindad.....	436
	Agrupación de teselas previa.....	437
	Comandos con funciones de vecindad predefinidas	439
4.9.	Rasterización.....	441
4.10.	Análisis estadístico zonal	442
4.11.	Intersección.....	443
	Intersección vectorial-raster	443
	Intersección raster-raster	445
H	Extensiones.....	449
1.	PGROUTING.....	451
	Instalación	451
	Caminos más cortos	452
	Topología de red	453
	Carga cartografía y topología	453
	Grafos directos, indirectos y costes	455
	Camino más corto (Dijkstra, A Star y restricciones de giro).....	455
	Utilización de datos OSM	461
	Utilización de cartografía sin estructura de red	466

2.	TOPOLOGÍA PERSISTENTE.....	470
2.1.	<i>Introducción</i>	470
	Modelo espagueti	470
	Modelo topología SQL/MM.....	471
2.2.	<i>Creación de topología</i>	472
	Ejemplo: topo-geometrías y primitivas	473
	Instalación	474
	Capa de topología.....	475
	Creación de primitivas topológicas	475
	Modificación de primitivas topológicas	480
	Resumen de la topología	482
	Acceso a primitivas topológicas.....	483
2.3.	<i>Capas de topo-geometrías</i>	483
	Creación de la capa	483
	Inserción de topo-geometrías	485
	Borrado de una capa de topo-geometrías	488
	Conversión a geometrías	488
	Visualización de capas de topo-geometrías.....	489
2.4.	<i>Creación automática de topo-geometrías</i>	491
2.5.	<i>Capas derivadas o hijas</i>	493
2.6.	<i>Análisis espacial</i>	494
	Simplificación de geometrías adyacentes.....	495
	Utilización de índices espaciales con topo-geometrías.....	496
I	<i>Anexos</i>	499
1.	NOTAS SOBRE ADMINISTRACIÓN	501
1.1.	<i>Variables de sistema</i>	501
	Modificación a nivel de sesión (show y set).....	501
	Modificación a nivel de sesión (vista pg_settings).....	502
	Modificación a nivel de base de datos.....	503
	Modificación de los valores por defecto.....	504
1.2.	<i>Autenticación del cliente</i>	504
	Recuperación de la contraseña de administración	507
1.3.	<i>Cluster de la base de datos</i>	507
	Arrancar y parar el servidor de PostgreSQL	509
	Inicializar y utilizar un cluster alternativo.....	510
1.4.	<i>Localización</i>	511
	Codificaciones soportadas	513
1.5.	<i>Vacuum</i>	514
	Comando Vacuum	515
	Comando Analyze.....	516
	Autovacuum.....	517
1.6.	<i>Roles de la base de datos y privilegios</i>	517
	Atributos de los roles.....	517
	Grupos de roles	519
	Grant y Revoke.....	520
1.7.	<i>Ficheros log</i>	523
1.8.	<i>Consumo de recursos</i>	528
1.9.	<i>Otros</i>	530

2.	SQL.....	531
2.1.	Lenguaje SQL.....	531
2.2.	Instrucciones SQL.....	532
2.3.	Definición de datos	533
	Listado de los ejemplos	533
	Tablas.....	533
	Uso de mayúsculas	535
	Dominios.....	535
	Esquemas	537
	Restricciones (not null, unique, primary key, check)	538
	Restricción de clave ajena o integridad referencial	543
	Programas de diseño conceptual	547
2.4.	Manipulación de datos.....	547
	Inserción, borrado y actualización.....	548
	Consultas elementales sobre una tabla	550
	Funciones de valor y agregadas.....	554
2.5.	Consultas sobre varias tablas	556
	Subconsultas (predicados In, All, Any, Exists, Distinct, Unique, ...)	556
2.6.	Trabajo con varias tablas (concatenaciones)	559
	Los operadores conjuntistas: Union, Except e Intersect	559
	Concatenaciones (Joins): Cross Join, Inner Join y Outer Join	560
2.7.	Inserción de filas provenientes de una consulta.....	565
	Creación de una tabla nueva.....	565
	Inserción de registros en una tabla existente	566
2.8.	Vistas.....	566
2.9.	Índices.....	567
	Creación y borrado	568
3.	SOLUCIONES	570
3.1.	Capítulo B.....	570
3.2.	Capítulo C.....	577
3.3.	Capítulo D.....	585
3.4.	Capítulo E.....	588
3.5.	Capítulo G.....	594
3.6.	Capítulo I.....	597
J	Recursos y bibliografía.....	599
1.	RECURSOS PRÁCTICOS Y TUTORIALES	601
2.	BIBLIOGRAFÍA.....	602

A Introducción

Este capítulo es meramente introductorio donde se sitúa al lector dentro del marco normativo seguido por PostGIS y se describen los estándares que serán referenciados a lo largo de la publicación, cuyo estudio es necesario para comprender el funcionamiento de una base de datos espacial.

Se repasa el proceso de instalación de PostgreSQL y PostGIS en máquinas con *MS Windows*, y se comenta brevemente el proceso en máquinas *Linux*. Tras ello, se introducen dos clientes SQL, uno de texto *psql* y otro gráfico *pgadmin*, centrándose en los comandos y utilización del cliente *psql*.

En cuanto al propio lenguaje SQL, se enumeran únicamente los tipos de datos más utilizados en PostgreSQL y se describe el proceso de creación de una base de datos nueva. Se hace hincapié en que el lector evalúe sus conocimientos de SQL y si lo estima conveniente lea y realice detenidamente la teoría y los ejemplos propuestos en el apartado I 2, pág. 531, donde se describe dicho lenguaje antes de seguir avanzando en la publicación y meterse de lleno en el mundo PostGIS.

1. Primeros pasos

PostGIS es una extensión de la base de datos PostgreSQL¹ que permite gestionar objetos geográficos. PostGIS añade la capacidad de utilizar PostgreSQL como base de datos espacial independiente o como repositorio de un Sistema de Información Geográfica² (SIG). Fue desarrollado por *Refractions Research*³. Actualmente, es un proyecto de la fundación de software libre geoespacial OSGeo⁴.

La versión 0.1 data del año 2001. La versión estable en el momento de escribir esta publicación es la 3.5 de septiembre de 2024. Cada varios meses se lanza una nueva versión que generalmente incorpora mejoras y correcciones de errores.

Desde su versión 1.1.0, PostGIS incorpora un modelo inicial de topología persistente, parecido a la estructura arco-nodo de *ArcInfo WorkStation*⁵, siguiendo las pautas marcadas por *Oracle Spatial* y su modelo de topología persistente⁶, disponible desde hace ya bastantes años. Este modelo de topología es actualmente una solución bastante madura, aunque quizás aún no demasiado conveniente para ser aplicado en un entorno de producción, sobre todo debido al excesivo tiempo empleado en la creación y procesado de los elementos topológicos.

Existen diferentes programas libres que permiten visualizar datos almacenados en PostGIS, además de otros formatos. Algunos de ellos son *gvSIG*, *QGIS*, *uDIG*, *OpenJUMP* o *Kosmo*, cuyas características se describen en el apartado B 3.2, pág. 69.

PostGIS está desarrollado en los lenguajes *C*, *C++* y *PL/PgSQL* (lenguaje procedural de PostgreSQL) y utiliza, entre otras, las bibliotecas de software libre *GEOS*⁷ (análisis espacial), *SFCGAL*⁸ (proceso de geometrías 3D) y *Proj4*⁹ (proyecciones). PostGIS utiliza *GEOS* para realizar la mayoría de los cálculos geométricos (tipos de objetos geométricos, relaciones y operaciones de análisis espacial entre geometrías 2D, etc.).

Como mejoras en las últimas versiones de PostGIS cabe destacar que a partir de la versión 1.5, es posible realizar ciertas operaciones espaciales con geometrías definidas en sistemas

¹ PostgreSQL es un Sistema Gestor de Bases de Datos Relacional (SGBDR) de código abierto disponible para múltiples plataformas, <https://www.postgresql.org/>.

² Un SIG es un sistema diseñado para la captura, almacenamiento, gestión, análisis y presentación de datos geográficos georreferenciados. Un SIG combina diferentes técnicas como cartografía, análisis estadístico y bases de datos.

³ <http://www.refractions.net/>

⁴ The Open Source GeoSpatial Foundation: <https://www.osgeo.org/>

⁵ *ArcInfo Workstation* fue uno de los programas pioneros en SIG; su primera versión data del año 1982, y no fue hasta la versión 7.1 cuando migró a sistemas operativos *MS Windows*. Llamado a desaparecer, posiblemente debido a la dificultad en su uso, implementa un modelo de topología persistente envidiado por muchos otros programas de SIG.

⁶ La topología persistente o explícita plasma las relaciones espaciales entre las geometrías de una capa o varias capas de forma física, en campos de atributos de tablas, tablas internas o cualquier otro mecanismo. Dichas relaciones espaciales se mantienen y sincronizan automáticamente con las entidades geométricas. Un sistema que soporte topología persistente generalmente implementa también un modelo de tolerancias de coincidencia de las coordenadas de las geometrías.

⁷ La biblioteca *GEOS*, <https://libgeos.org/>, es una migración al lenguaje *C++* de la biblioteca *JTS (Java Topology Suite)*, <https://www.osgeo.org/projects/jts/> desarrollada en *Java*.

⁸ La biblioteca *SFCGAL* (<https://sfcgal.gitlab.io/SFCGAL/>) es un envoltorio en *C++* de la biblioteca *CGAL* (<https://www.cgal.org/>) y proporciona operaciones 3D siguiendo la especificación *Simple Feature Access* del OGC.

⁹ <https://proj.org/>

de referencia no proyectados (sistemas de coordenadas geográficas), es decir, se puede calcular, por ejemplo, la distancia entre dos puntos sobre un determinado elipsoide de referencia. Para ello, se introdujo un nuevo tipo de datos llamado *geography*, además del tipo de datos *geometry*; este último es el que se sigue utilizando para realizar análisis espaciales completos utilizando sistemas proyectados.

A partir de la versión 2.0 de PostGIS se incluye soporte de datos *raster* mediante la integración de una extensión *raster* incluida en la versión oficial. De esta forma, PostGIS 2.0 puede almacenar y gestionar la información *raster* en tablas como si de información vectorial se tratara y realizar operaciones de análisis espacial *raster* o mixtas entre ambos modelos.

Otras mejoras importantes durante los últimos años han sido el soporte de indexación espacial en tres dimensiones, la ampliación de funcionalidades 3D (especialmente a partir de la utilización de la biblioteca SFCGAL) y la extensión de topología persistente.

1.1. Normativa relacionada

A lo largo de esta publicación se hace referencia a algunos documentos publicados por el *Open Geospatial Consortium*¹, en adelante el OGC.

El primero de estos documentos es la especificación del OGC titulada “*OpenGIS Simple Features Implementation Specification for SQL. Versión 1.1.*” (en adelante lo denominaremos SFS1.1 para abreviar). Esta especificación fue publicada en el año 1999 y tiene como objetivo definir un esquema SQL que soporte el acceso, almacenamiento, consulta y actualización de fenómenos geoespaciales simples.

PostGIS y algunas bibliotecas tan importantes, como *JTS* y *GeoTools*², están apoyadas principalmente en esta especificación. Algunas limitaciones de esta primera versión son la utilización únicamente de coordenadas X e Y (las coordenadas Z y M no se tuvieron en cuenta) y la estandarización de las firmas de los métodos propuestos.

Unos años después, dicho estándar fue revisado por el OGC, quien lo dividió en dos documentos, obteniendo así las normas ISO 19125-2:2004 e ISO 19125-1:2004, aunque el contenido permanece prácticamente el mismo que la especificación SFS1.1 original. En adelante, denominaremos ambas normas SFA1.1.0:

- La Norma ISO 19125-2:2004 también se conoce como “*OpenGIS Implementation Specification for Geographic information – Simple Feature Access – Part 1: Common Architecture. Versión 1.1.0*”, y describe el modelo de geometrías utilizado y la forma de representación de las geometrías (WKT, WKB) y de los sistemas de referencia (WKT).
- La Norma ISO 19125-1:2004 se conoce como “*OpenGIS Implementation Specification for Geographic Information – Simple Feature Access – Part 2: SQL Option. Versión 1.1.0*”, y describe, entre otros aspectos, el esquema SQL para soportar almacenamiento, consulta y actualización de objetos espaciales basados en geometrías de dos dimensiones.

¹ El *Open Geospatial Consortium, Inc* (OGC) es un consorcio internacional de más de 400 compañías, instituciones y universidades con el objetivo de desarrollar especificaciones y estándares que faciliten soluciones interoperables para la consulta, creación y gestión de la información geográfica.

² Biblioteca desarrollada en Java, utilizada intensamente en *GeoServer* (servidor de cartografía). Además de estar integrada con la biblioteca *JTS*, añade funcionalidades como el *renderizado* de geometrías y el acceso a datos espaciales, utilizando diferentes *drivers* como PostGIS, *ShapeFile*, etc. <https://geotools.org/>

Por último, en el año 2006 (corrigendum del 2011) apareció una nueva versión (1.2.1) de la especificación SFA, en adelante SFA1.2.1:

- “*OpenGIS Implementation Specification for Geographic information – Simple feature access – Part 1: Common architecture*”. Versión 1.2.1.
- “*OpenGIS Implementation Specification for Geographic information – Simple feature access – Part 2: SQL option*”. Versión 1.2.1.

Entre algunas novedades importantes de estas normas encontramos:

- Inclusión de las coordenadas Z y M en los tipos de geometrías y en las definiciones WKT y WKB.
- Inclusión de anotaciones para definir textos (topónimos).
- Nuevos métodos de referencia lineal de geometrías.
- Inclusión de orientación en los elementos poligonales.
- Nuevos tipos de geometrías para almacenar TINs y superficies poliédricas.
- No se estandarizan los nombres de las firmas de los métodos.

Paralelamente a estas especificaciones y normas, dentro del estándar SQL se incluye la norma ISO 13249-3:2004 (actualmente 13249-3:2016) “*Information Technology – Database Languages – SQL Multimedia and Application Packages – Part 3: Spatial*”, o también denominada SQL/MM (SQL Multimedia). Este documento no tiene de momento equivalente en el OGC (se debe adquirir previo pago a la organización ISO). Es una norma más evolucionada que SFS1.1, SFA1.1.0 e incluso SFA1.2.1, de la cual las bases de datos espaciales, PostGIS entre ellas, han adoptado ciertas características. Esto ha originado una confusión sobre los estándares que sigue PostGIS y/u otras bases de datos espaciales. El OGC está tratando de corregir esta situación mediante una nueva especificación que busca, en un futuro, ser un nexo de unión con la norma SQL/MM. En las versiones más recientes, puede decirse que PostGIS está tratando de seguir más la norma SQL/MM que la última versión de las normas SFA.

Es importante mencionar que PostGIS no implementa completamente el estándar SFA1.2.1 ni el SQL/MM. Por ejemplo, el comando `ST_Astext`, que se estudiará más adelante, no soporta anotaciones, orientación en los elementos poligonales, etc.

Por todo ello, aún no puede afirmarse que PostGIS cumpla completamente con SFA1.2.1 y/o SQL/MM, aunque es posiblemente la base de datos espacial que más se aproxima a dichos estándares.

1.2. Instalación

PostgreSQL

Aunque PostgreSQL está disponible para múltiples plataformas, en este apartado vamos a instalar la versión para *MS Windows*, no porque la instalación sea más sencilla que en sistemas *Linux*, sino porque dicha instalación depende de la distribución de *Linux* utilizada. De todas formas, los usuarios de *Linux* pueden instalar PostgreSQL obteniendo los binarios desde la página oficial o utilizando los correspondientes gestores de paquetes correspondientes a cada distribución.

En muchos casos, el procedimiento de instalación en *Linux* es incluso más sencillo que en el propio *MS Windows*.

Hay que comentar que todos los comandos, tanto SQL como los métodos PostGIS, que veremos en este libro, son independientes del sistema operativo que se esté utilizando. El lector puede, por lo tanto, elegir su sistema operativo favorito e instalar el software utilizado en este libro sin problemas.

Para los ejemplos, no se necesita ninguna extensión o configuración especial de PostgreSQL. La instalación de esta forma se simplifica aún más, como aparece a continuación.

1.- Descarga de los binarios desde la página oficial de PostgreSQL¹. En dicha página web, en la sección *downloads*, en el párrafo '*Binary packages*', seleccionamos *Windows*. A continuación, haz clic en *download the installer* y elige la versión deseada de PostgreSQL y del sistema operativo (*p. ej.*: para *MS Windows* sería *Win x86-64*).

En función de la versión de *MS Windows* instalada, las posibles combinaciones son:

- *MS Windows* 64 bits -> PostgreSQL 64 bits -> PostGIS 64 bits.
- *MS Windows* 64 bits -> PostgreSQL 32 bits -> PostGIS 32 bits.
- *MS Windows* 32 bits -> PostgreSQL 32 bits -> PostGIS 32 bits.

2.- Tras ejecutar el fichero (*postgresql-17.x-x-windows-x64.exe* o similar), se iniciará el proceso de instalación y aparecerá la pantalla de bienvenida.

Presionamos siguiente.

Directorio de instalación: introducimos el directorio de instalación. Por ejemplo:
C:\PostgreSQL\17

Presionamos siguiente.

Selección de Componentes: deja marcados todos los componentes.

Directorio de datos: Introducimos el directorio donde se almacenarán los datos. El instalador nos da la posibilidad de almacenar los datos en otro directorio o incluso en otra unidad física. En nuestro caso, los dejaremos en el directorio *data*, dentro del directorio de instalación principal de PostgreSQL: C:\PostgreSQL\17\data

Presionamos siguiente.

Contraseña: introducimos la nueva contraseña del administrador de la base de datos. Esta contraseña también será utilizada para establecer el servicio de *MS Windows* que arrancará el servidor PostgreSQL.

Presionamos siguiente.

Puerto: el puerto de comunicación de PostgreSQL por defecto es 5432. En el caso de que este puerto ya esté ocupado o que queramos cambiar el puerto del servidor, este es un buen momento para hacerlo. Dejamos el valor por defecto: 5432.

Presionamos siguiente.

Opciones avanzadas: en este paso hay que introducir la configuración regional de PostgreSQL. Esta configuración hace referencia a los formatos de fecha y lenguaje utilizados en los mensajes del servidor, etc. Lo dejamos en '*[Configuración Regional por defecto]*'.

Resumen de la instalación: presionamos siguiente.

¹ <https://www.postgresql.org/download/>

Listo para instalar: presionamos siguiente, y comenzará el proceso de instalación. El programa se instalará rápidamente e inicializará también el *cluster* de datos (estructura de ficheros necesaria para almacenar todas las bases de datos) en el directorio `C:\PostgreSQL\17\data`. Hay que estar atentos porque el antivirus del ordenador puede tratar de bloquear el acceso al puerto 5432.

Stack Builder: Tras la instalación aparecerá una última pantalla donde se pregunta si se quiere ejecutar el programa *Stack Builder*. *Stack Builder* es un programa que permite instalar complementos para PostgreSQL mediante una conexión a Internet. Desactivamos la casilla, más adelante se puede ejecutar de nuevo si es necesario.

Comprobación del servicio

Una primera comprobación necesaria, pero no suficiente, de que el servidor está funcionando correctamente, consiste en asegurarse de que el servicio de PostgreSQL ha sido iniciado.

Panel de Control > Herramientas administrativas > Servicios.

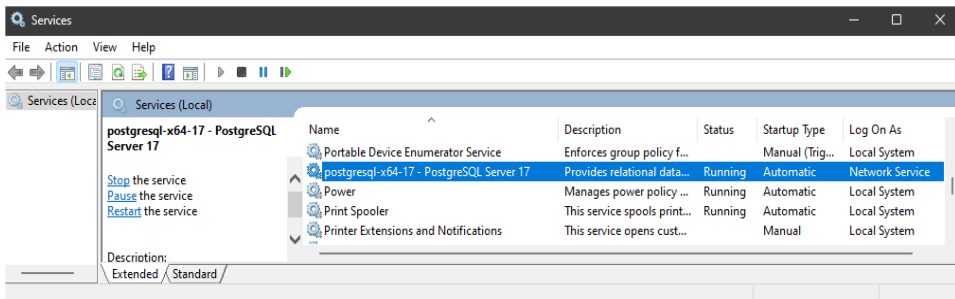


Figura 1 Servicio de PostgreSQL en MS Windows

Desde esta misma ventana, con un doble clic, sobre el servicio aparecerá una segunda caja de diálogo donde se puede detener e iniciar el servicio y consultar información sobre el servicio.

Path del sistema

A lo largo de esta publicación, se usan algunas utilidades de PostgreSQL que se ejecutan como comandos dentro de una terminal o símbolo del sistema (ver Tabla 1, pág. 31). Estos comandos en *MS Windows* se copian en el proceso de instalación de PostgreSQL dentro del directorio *bin* y, para acceder a ellos desde cualquier directorio, es necesario configurar el *path* o ruta del sistema de búsqueda de binarios.

Para ello, en la barra de tareas de MS Windows, busca el programa ‘entorno’ y selecciona ‘Editar las variables de entorno del sistema’. Presiona el botón ‘Variables de entorno’.

Haz clic en la variable *Path* (en la caja de variables del sistema) y edítala para modificarla. Añade la ruta del directorio *bin* de PostgreSQL (imagen inferior izquierda). Si estás en una versión anterior a MS Windows 10, deberás añadir la ruta al final de la cadena de texto, separada por un punto y coma (imagen inferior derecha).

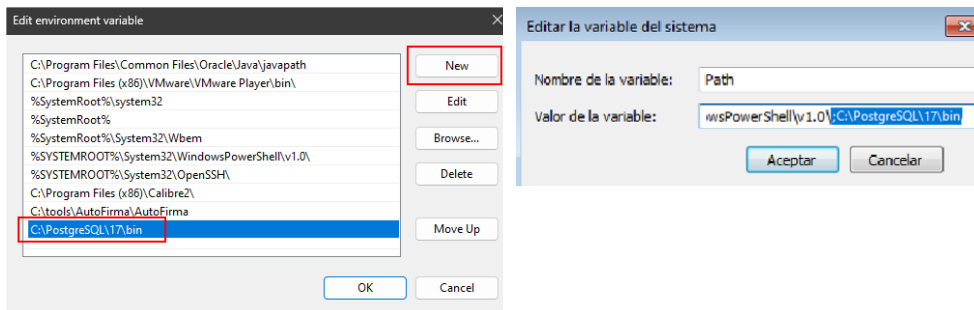


Figura 2 Path del sistema en MS Windows

Instalación de PostGIS

La instalación de PostGIS se puede realizar de forma automática en *MS Windows* utilizando el programa *Stack Builder* accesible desde el menú de inicio / PostgreSQL. En distribuciones *Linux* u otros sistemas operativos habrá que utilizar el gestor de software o de paquetes propio de cada distribución, siendo generalmente también un proceso bastante sencillo.

La instalación mediante *Stack Builder* en *MS Windows* consta de los siguientes pasos:

- Ejecución de *Stack Builder* desde el menú de inicio / PostgreSQL.
- Selecciona en el desplegable el servidor recién instalado (p. ej. *PostgreSQL 17 (x64 on port 5432)*)

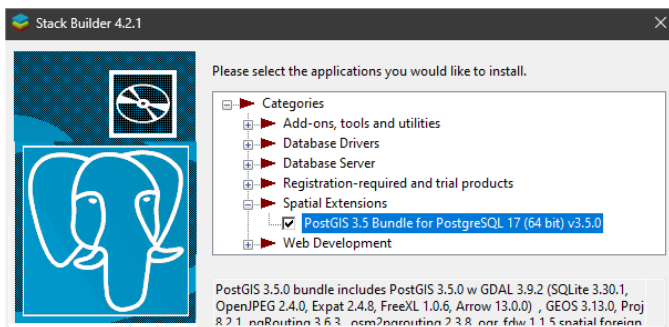


Figura 3 Instalación de PostGIS con *Stack Builder*

- Selecciona el paquete PostGIS dentro del apartado *Spatial Extensions* (la elección de 32 o 64 bits debe coincidir con la versión instalada de PostgreSQL).
- En la siguiente pantalla introducimos un directorio temporal donde el instalable de PostGIS será descargado.
- A continuación, *Stack Builder* lanzará el instalable de PostGIS y aparecerá la pantalla de aceptación de licencia de PostGIS.

Si hay problemas en la descarga, siempre es posible descargar el instalable de PostGIS directamente desde: <https://download.osgeo.org/postgis/windows/> y ejecutarlo. Tras ello, continuaremos igual que con el *Stack Builder*.

- En la siguiente ventana aparecerán los componentes de PostGIS a instalar: el propio PostGIS y una base de datos espacial de ejemplo. Dejamos las opciones marcadas por defecto. Asegúrate que la opción “crear base de datos espacial” está desmarcada, ya que la crearemos a posteriori para entender mejor el proceso.
- En la siguiente pantalla seleccionamos el directorio de instalación de PostgreSQL. Generalmente, el instalador detectará dicho directorio, pero, si no fuera así, debe seleccionarse de forma manual: `C:\PostgreSQL\17`
- Al finalizar nos pedirá que aceptemos la creación de varias variables de entorno, respondemos que sí a las tres preguntas.

1.3. Clientes SQL

Tras la instalación de PostgreSQL nuestro ordenador dispondrá de dos clientes para acceder a PostgreSQL: un cliente de texto *psql* y un cliente gráfico *pgadmin*. Si se está utilizando *Linux* es posible que el software *pgadmin* deba instalarse aparte.

Aunque el cliente gráfico *pgadmin* es más amigable que el cliente de texto *psql*, por el momento y con fines didácticos, es mejor empezar a manejar el cliente de texto *psql*. Más adelante, cuando el lector tenga un conocimiento más profundo de la base de datos, podrá utilizar el cliente gráfico si así lo estima conveniente.

Clientes de texto: *psql*

Psql es un cliente de línea de comandos distribuido junto a PostgreSQL. También se le conoce como monitor o terminal interactivo de PostgreSQL.

Con *psql*, se dispone de una simple pero potente interfaz para comunicarse con el servidor PostgreSQL. Una vez establecida una sesión con el cliente *psql*, podemos empezar a introducir las sentencias SQL.

Antes de empezar a introducir SQL a través del cliente *psql*, comentamos algunas notas:

- PostgreSQL **convierte a minúsculas** todas las cadenas que no se encuentren entre comillas (al contrario que el estándar SQL92).

De la misma forma, si los nombres de las tablas o los nombres de los campos de las tablas no se introducen entre comillas, serán convertidos a minúsculas. Como regla general en esta publicación y para simplificar el código SQL, utilizaremos los nombres de tablas y campos sin usar comillas; es decir, PostgreSQL los interpretará siempre en minúsculas.

- Valor nulo: representado por *NULL*.
- Un literal de cadena de caracteres va encerrado entre comillas simples: *'cadena'*.
- Cada sentencia SQL debe ser terminada con un punto y coma.
- Hay dos tipos de comentarios en PostgreSQL: comentario de una sola línea, usando `'-'`, y comentarios de bloque, usando `'/*'` para iniciar el bloque y `'*/'` para finalizarlo.

Ejemplo:

```
-- Esto es un comentario de línea
/* Esto
Es un comentario
De bloque */
```

Codificación terminal (MS Windows)

En sistemas operativos *MS Windows* es necesario configurar la codificación de caracteres utilizada por la consola o símbolo del sistema para su utilización con el terminal *psql*:

1. Abrimos una consola o símbolo del sistema (programa `cmd` en MS Windows).
2. Puedes probar a cambiar la fuente de la consola a otra que quizás se vea mejor, para ello seleccionamos propiedades del menú principal de la ventana y en la pestaña fuentes cambiamos la fuente a “*Lucida Console*”.
3. Utilizamos el comando `chcp` para cambiar la codificación de los caracteres:

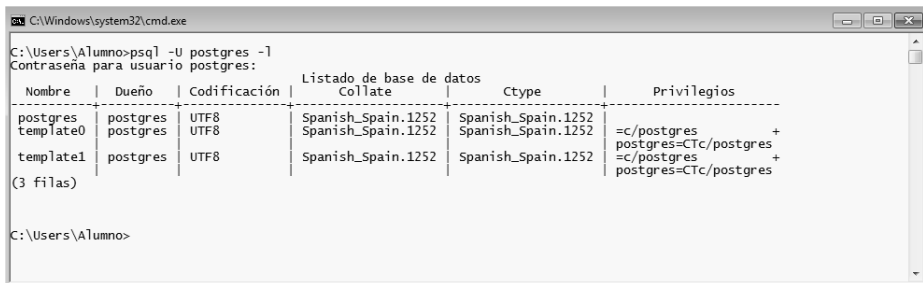
```
consola> chcp 1252
```

Ahora ya podemos ejecutar el comando *psql* de forma adecuada tal como se indica en el siguiente apartado.

Conexión al servidor

Primero vamos a ver qué bases de datos hay en el sistema (en una instalación limpia de PostgreSQL deberíamos tener al menos las bases de datos *template0* y *template1*).

Para ejecutar el comando *psql*, abrimos una consola del sistema, ‘Símbolo del sistema’:



```
C:\Windows\system32\cmd.exe
C:\Users\Alumno>psql -U postgres -l
Contraseña para usuario postgres:
Listado de base de datos
-----
```

Nombre	Dueño	Codificación	Collate	Ctype	Privilegios
postgres	postgres	UTF8	Spanish_Spain.1252	Spanish_Spain.1252	
template0	postgres	UTF8	Spanish_Spain.1252	Spanish_Spain.1252	=c/postgres + postgres=CTc/postgres
template1	postgres	UTF8	Spanish_Spain.1252	Spanish_Spain.1252	=c/postgres + postgres=CTc/postgres

```
(3 filas)

C:\Users\Alumno>
```

Figura 4 Listado bases de datos

Listado de las bases de datos que hay en el sistema y sus propietarios:

```
consola> psql -U postgres -l
```

Donde *postgres* se corresponde con el usuario con el cual queremos acceder al sistema.

En nuestro caso el usuario *postgres* es el administrador del sistema y su contraseña es la que se introdujo en el proceso de instalación de PostgreSQL.

Una vez vistas las bases de datos que hay actualmente en el sistema podemos autenticarnos y acceder a una de ellas utilizando la opción ‘-d’, por ejemplo, *template1*.

```
consola> psql -U postgres -d template1
```

Tras introducir la contraseña, el *prompt* del sistema cambia de aspecto lo que nos indica que se ha accedido a la base de datos *template1* de forma correcta.

```
template1=#
```

En este momento el cliente *psql* quedará a la espera de la introducción de órdenes del usuario. Órdenes que deben ser instrucciones SQL o comandos del propio cliente *psql*; es decir, desde este *prompt* no se pueden ejecutar comandos del sistema operativo. Para ello, habrá que abrir otra consola del sistema o salir del *prompt* de *psql* con la orden ‘\q’.

Los comandos de utilidad de PostgreSQL se instalan en el directorio *bin* de PostgreSQL en *MS Windows* y en */usr/bin* en *Linux*, y su funcionamiento es similar en cualquier sistema operativo. Los más utilizados y que se verán en este libro son:

Algunos comandos PostgreSQL del sistema operativo	Descripción
<code>psql</code>	Cliente SQL de texto de PostgreSQL.
<code>createdb</code>	Creación de una nueva base de datos.
<code>dropdb</code>	Borrado de una base de datos existente.
<code>createlang</code>	Creación de un nuevo lenguaje en la base de datos. En desuso, se utiliza en su lugar <i>create extension</i> en SQL.
<code>pg_dump</code>	Exportación de la base de datos.
<code>pg_vacuum</code>	Limpia las bases de datos y optimiza el espacio ocupado.

Tabla 1 Comandos de PostgreSQL

Desde el cliente *psql*, se puede ejecutar tanto comandos SQL como comandos *psql*. Algunos de los comandos *psql* son:

Comando psql	Descripción
<code>\?</code>	Ayuda de los comandos de <i>psql</i> como los comentados en esta tabla.
<code>\help</code> o <code>\h</code>	Ayuda comandos SQL de PostgreSQL. Ejemplo: <code>\h CREATE TABLE</code>
<code>\q</code>	Sale y vuelve al <i>shell</i> .
<code>\l</code>	Lista todas las bases de datos.
<code>\d</code>	Lista todas las tablas, secuencias y vistas.
<code>\d [nombre]</code>	Lista detallada de las tablas, secuencias, vistas o índices, describiendo cada uno de sus campos o parámetros.
<code>\dD</code>	Lista los dominios.
<code>\dn</code>	Lista los esquemas.
<code>\a</code>	Cambia el formato de la salida del texto, alineado o sin alinear.
<code>\x</code>	Cambia el formato a lista, en lugar de tabla
<code>\i</code>	Ejecuta un fichero de texto con comandos SQL.
<code>\x on</code> <code>off</code>	Activa / Desactiva el modo expandido de visualización de las tablas

Tabla 2 Comandos del cliente *psql*

Ejemplos de uso de comandos *psql* sobre la base de datos *template1*:

```
template1=# \l
template1=# \?
template1=# \q
```

Para salir del intérprete de comandos *psql* y volver a la consola inicial, se utiliza el comando `\q`.

Cientes gráficos: *pgadmin*

En esta publicación se insiste en utilizar la base de datos PostgreSQL desde un terminal de texto como es *psql*, para que el usuario conozca las sentencias SQL de una forma más didáctica. En el mercado existen muchas aplicaciones tanto *open source* como comerciales que actúan como interfaces de usuario gráficas¹ (clientes gráficos) para interactuar con la base de datos de una forma más amigable que desde la consola del sistema mediante comandos SQL, especialmente a la hora de introducir y consultar tablas de datos.

Quizás las aplicaciones más conocidas de PostgreSQL (ambas *open source*) son *pgAdmin*² y *phpPgAdmin*³.

En *MS Windows*, la aplicación *pgAdmin* se instala por defecto al instalar PostgreSQL. En *Linux* no se instala por defecto y, generalmente, será necesario instalar la aplicación desde el gestor de *software*/paquetes de cada distribución, pero es una tarea sencilla. En cuanto a *phpPgAdmin*, está enfocado a un uso dentro de una arquitectura cliente-servidor web de forma que cualquier usuario puede acceder al servidor de PostgreSQL a través de la web. Resulta más complicado de instalar y ofrece menos características que *pgAdmin*, aunque es preferible en el caso de necesitar una interfaz web de administración.

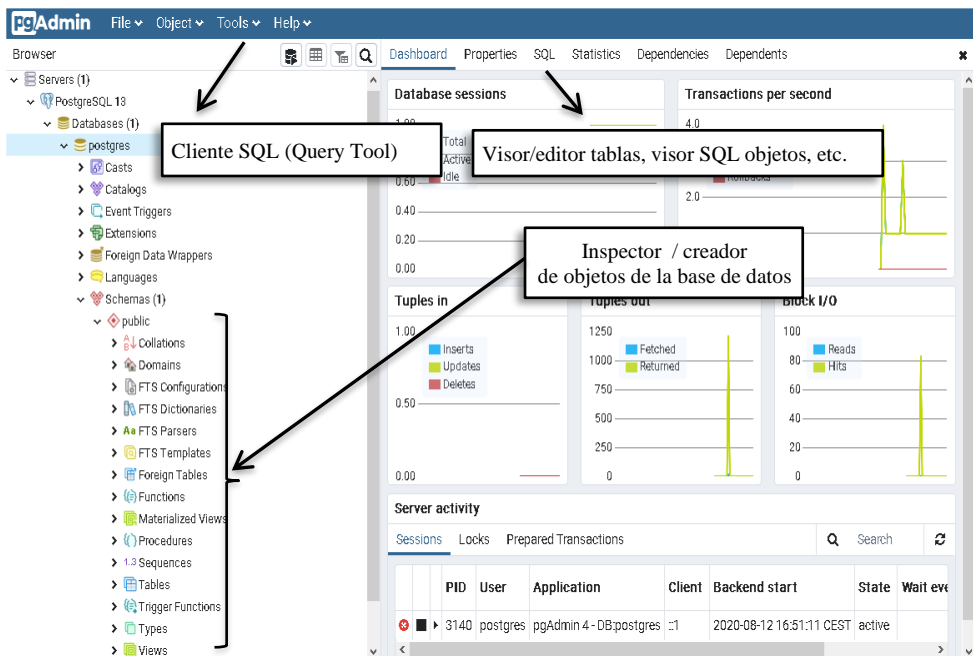


Figura 5 Herramienta *pgAdmin* para PostgreSQL

¹ https://wiki.postgresql.org/wiki/Community_Guide_to_PostgreSQL_GUI_Tools

² Herramienta multi-plataforma de administración de bases de datos PostgreSQL. <https://www.pgadmin.org>

³ Herramienta de administración de PostgreSQL. <https://phpPgAdmin.sourceforge.net/>

Algunas de las funcionalidades de *pgAdmin* son las siguientes:

- Inspección, creación y borrado de objetos de la base de datos: tablas, dominios, vistas, secuencias, esquemas, funciones, disparadores, etc.
- Cliente/Editor SQL parecido al cliente *psql* (ver Figura 8).
- Acceso a la consola *psql*.
- Herramientas de administración: copias de seguridad (*pg_dump*), *explain analyze* de forma gráfica (ver Figura 32, pág. 109) etc.

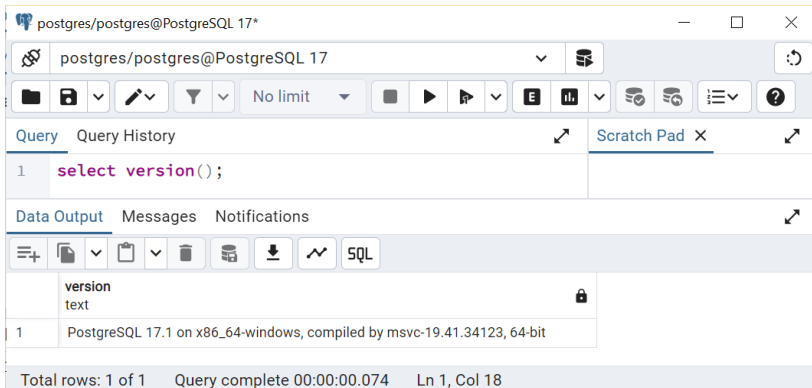


Figura 6 Cliente SQL en *pgAdmin4*

1.4. Tipos básicos de datos

En la siguiente tabla se muestran algunos tipos de datos de PostgreSQL (los más utilizados y los que veremos en esta publicación). Aunque en PostgreSQL muchos de estos tipos de datos disponen de otros nombres auxiliares o alias (p. ej.: un tipo de datos *smallint* se puede denominar también como *int2*, un *integer* como *int4*, un *real* como *float4* o un *double precisión* como *float8*, etc.), en esta publicación se trata de utilizar el nombre compatible SQL para favorecer la portabilidad a otros SGBD.

En la tercera columna de la tabla se hace referencia a la versión del primer estándar SQL que soportó dicho tipo de dato.

Tipo de datos	Descripción	Estándar
Caracteres		
varchar (n)	Cadena de caracteres variable, limitada a <i>n</i> caracteres. También conocido como <i>character varying (n)</i> .	SQL92
char (n)	Cadena de caracteres de longitud fija de <i>n</i> caracteres. Defecto <i>n=1</i> . También conocido como <i>character (n)</i> .	SQL89
Númericos enteros		
boolean	Booleano <i>TRUE / FALSE</i> .	SQL99
smallint	Entero de 2 <i>bytes</i> con signo. Alias en PostgreSQL: int2	SQL89
integer	Entero de 4 <i>bytes</i> con signo. Alias en PostgreSQL: int4	SQL92
bigint	Entero de 8 <i>bytes</i> con signo. Alias en PostgreSQL: int8	SQL2003

Numéricos decimales		
real	Flotante de 4 <i>bytes</i> según <i>IEEE 754</i> . Alias en PostgreSQL: float4	SQL89
double precision	Flotante de 8 <i>bytes</i> según <i>IEEE 754</i> . Alias en PostgreSQL: float8	SQL89
numeric(p,s)	Tipo exacto de <i>P</i> dígitos (<i>Precision</i>) y <i>S</i> decimales (<i>Scale</i>). También conocido como <i>decimal (p, s)</i> .	SQL99
Fechas y hora		
timestamp (n)	Fecha / hora (sin zona horaria), <i>n</i> dígitos / fracción de seg.	SQL92
time (n)	Hora (sin zona horaria), con <i>n</i> dígitos / fracción de seg.	SQL92
date	Fecha.	SQL92
interval (n)	Intervalo hora, con <i>n</i> dígitos de fracción de seg.	SQL92
Específicos de PostgreSQL		
text	Cadena de caracteres de longitud variable sin limitación. Este tipo es similar al tipo <i>Varchar</i> .	PostgreSQL
serial	Entero de 4 <i>bytes</i> autoincremental (secuencia), <i>bigserial</i> utiliza un entero de 8 <i>bytes</i> .	PostgreSQL
oid	Identificador de objeto (4 <i>bytes</i>), utilizado en tablas internas (<i>pg_catalog</i>). No aconsejable para utilizarlo como identificador en tablas de usuario.	PostgreSQL
bytea	Secuencia de <i>bytes</i> . Conocido en el estándar SQL y en otras bases de datos como <i>BLOB</i> o <i>Binary Large Object</i> .	PostgreSQL
Específicos de PostGIS		
geometry	Geometrías en formato binario.	PostGIS
geography	Geometrías en coordenadas geográficas (WGS 84), para realizar análisis espacial sobre el esferoide.	PostGIS
raster	Teselas en formato <i>raster</i> .	PostGIS

Tabla 3 Tipos básicos de PostgreSQL

En la ayuda *online* de PostgreSQL¹, en el capítulo *Datatypes*, se puede encontrar una definición más amplia y detallada de todos los tipos que soporta PostgreSQL.

PostgreSQL también dispone de algunos alias para referirse de una forma más corta a ciertos tipos SQL como: *smallint (int2)*, *integer (int4)*, *bigint (int8)*, *real (float4)* o *double precision (float8)*.

En PostgreSQL los tipos para almacenar cadenas de caracteres de longitud fija, *Char(n)* y *Character(n)* son incluso más lentos que los tipos de longitud variable, además de ocupar más espacio de almacenamiento. Se aconseja el uso de *Varchar* o *Character Varying*. El parámetro del número de caracteres máximo (*n*) del tipo *Varchar(n)* es opcional.

¹ <https://www.postgresql.org/docs/current/datatype.html>

1.5. Notas antes de empezar

Creación de una nueva base de datos

Llegado a este punto vamos a crear nuestra primera base de datos con la cual empezaremos a realizar los ejemplos de este apartado. Para ello, utilizaremos el comando *createdb*, que como se ha comentado anteriormente, al ser un comando del sistema operativo, no se puede ejecutar dentro del cliente *psql* sino directamente desde una terminal o símbolo del sistema.

```
consola> createdb -U postgres b1
```

Este comando crea la base de datos *b1* cuyo dueño es el usuario *postgres* (al haberla creado). También se puede solicitar ayuda al sistema o borrar la base de datos creada con los siguientes ejemplos:

```
createdb --help
dropdb --help
dropdb -U postgres b1
```

Cambio de contraseña

Si se desea cambiar la contraseña de un usuario/role se puede utilizar la orden SQL *Alter User* desde un cliente *psql*, p. ej.: la siguiente sentencia cambia la contraseña del usuario/role *postgres* (los usuarios/roles y contraseñas son comunes a todas las bases de datos):

```
b1=# ALTER USER postgres PASSWORD 'nuevacontraseña' ;
```

Conocimientos previos

Llegado a este punto el lector debe evaluar sus conocimientos previos de SQL. En el caso de que no disponga de ellos es estrictamente necesario que antes de seguir leyendo esta publicación y adentrarse en el mundo PostGIS lea detenidamente el capítulo I 2, pág. 531, donde se realiza una introducción básica al lenguaje SQL de forma práctica.

La comprensión y utilización adecuada de PostGIS pasa por un conocimiento al menos básico-medio del lenguaje SQL.

Por el contrario, si el lector ya ha estudiado o utilizado el lenguaje SQL no está de más que repase de forma rápida los conceptos explicados en dicho capítulo, especialmente:

- Restricciones de tipo *check*, clave primaria, unicidad, etc.
- Agrupaciones y funciones agregadas.
- Concatenaciones internas y externas.
- Índices.
- Vistas.

2. Cartografía y código fuente de los ejemplos

Desde <https://cartosig.webs.upv.es> también se puede obtener el fichero *datoslibropostgis-xx.zip*, que contiene la cartografía y el código SQL utilizado en los ejemplos del libro. Al descomprimir el *zip* se obtienen dos carpetas:

codigo: contiene ficheros de texto *.sql* con todas las sentencias SQL y comandos ejecutados en los ejemplos, de esta forma el usuario puede ir copiando y pegando el código.

datos: contiene tanto la cartografía de los ejemplos (en varios formatos como *shape*, *arc-info*, *osm*, *raster* y sobre todo en formato SQL para su importación directa a PostGIS) como otros ficheros utilizados en el libro (ficheros *NTv2*, etc.).

B Núcleo

En este capítulo se describe una parte importante de la funcionalidad de PostGIS que es necesario conocer y manejar con soltura, lo que capacitará al lector para la realización de análisis espaciales más complejos como los expuestos en el siguiente capítulo. Se comienza detallando el proceso de creación de una base de datos espacial, las tablas de metadatos utilizadas por PostGIS y la gestión de tablas espaciales.

A continuación, se introducen los diferentes tipos de geometrías soportados por PostGIS, primero de una forma sencilla para que el lector pueda empezar a realizar sus primeras operaciones espaciales con PostGIS, y más adelante de forma detallada describiendo en profundidad el modelo de fenómenos simples o *simple features* del OGC en que se basa PostGIS. De esta forma se estudia cada uno de los tipos geométricos, cómo se forman, qué restricciones presentan, cómo se definen desde un punto algebraico, etc.

También se estudian diferentes procedimientos para importar y exportar datos cartográficos y tablas alfanuméricas en PostgreSQL / PostGIS, utilizando tanto los comandos de utilidad de PostGIS como otros programas externos interesantes, también *open source*, potentes como *GDAL/OGR*.

Es necesario que el lector conozca en profundidad el manejo de las relaciones espaciales entre las geometrías, aspecto que es utilizado continuamente en el análisis espacial no solo en PostGIS sino en cualquier base de datos espacial o SIG de escritorio, ya que los fundamentos teóricos introducidos por los estándares son similares en la mayoría de los software de SIG, para ello se estudia los predicados espaciales del OGC soportados por PostGIS tanto de forma gráfica como analítica tratando de resolver todas las dudas

La indexación espacial, cuya aplicación es crucial, se utiliza a partir de ahora en todos los ejemplos realizados en esta publicación.

Por último, el capítulo se cierra con la creación de tablas espaciales donde el lector puede almacenar los resultados de cualquier análisis espacial, y vistas espaciales muy útiles para realizar un control dinámico de la cartografía cuyo uso se mejora con las llamadas reglas para modificar registros en las vistas.

1. La Base de datos espacial

Tras instalar PostGIS en el ordenador aún, no se puede utilizar su funcionalidad en ninguna base de datos. Para ello, se necesita ‘espacializar’ o dotar a la base de datos que queramos utilizar con PostGIS de la funcionalidad espacial requerida.

‘Espacializar’ o convertir una base de datos en espacial consiste en añadir a una base de datos todos los procedimientos almacenados (funciones SQL) de PostGIS, así como algunos objetos como vistas que contienen un inventario de las tablas espaciales, tablas con los sistemas de referencia, etc. Estas funcionalidades se agregan mediante la ejecución sobre la base de datos que queremos ‘espacializar’ de miles de comandos SQL (que generalmente se agrupan en unos pocos ficheros de texto que vienen con la instalación de PostGIS). Este procedimiento se puede realizar de dos maneras diferentes en PostgreSQL:

1. Método sin utilizar *extensions*: Localizando los ficheros de texto que contienen los cientos de comandos SQL de PostGIS y ejecutando dichos ficheros SQL en la base de datos a convertir en espacial.
2. Método utilizando *extensions*: Ejecutando de forma muy sencilla únicamente varios comandos *create extension*. Este comando añade toda la funcionalidad de PostGIS de forma transparente, sin la necesidad de ejecutar ficheros SQL en la base de datos.

La forma más práctica, rápida y habitual es la segunda; es decir, **utilizando el comando *create extension***. La primera forma, aunque mucho más inusual, la veremos primero porque, aunque **no se suele utilizar**, sí que ayuda a comprender muy bien lo que significa convertir una base de datos en espacial.

Algunos de los comandos necesarios para convertir una base de datos en espacial requieren permisos de administrador de PostgreSQL. Por ello, para simplificar las cosas, utilizaremos el usuario *postgres* para conectarnos a la base de datos, ya que generalmente es administrador de PostgreSQL. Utilizaremos dicho usuario para realizar todos nuestros ejemplos de PostGIS, evitando problemas.

1.1. Creación base de datos espacial sin utilizar *extensions*

Módulo principal y vectorial

1. Localizar los ficheros *postgis.sql* y *spatial_ref_sys.sql* dentro de la instalación de PostGIS.

El primer paso consiste en localizar los ficheros *postgis.sql* y *spatial_ref_sys.sql* en nuestro ordenador. Estos ficheros han sido copiados en el proceso de instalación de PostGIS y su ubicación depende tanto del sistema operativo y su versión como de la versión de PostgreSQL y PostGIS instalada.

La ruta o *path* al fichero *postgis.sql* vendrá dada según el sistema operativo y la versión de PostgreSQL y PostGIS instalada. A continuación, se muestran los directorios por defecto utilizados por algunas distribuciones (ejemplo con PostgreSQL 17 y PostGIS 3.5):

- *OpenSuse*:

```
/usr/share/postgresql/contrib/postgis-3.5
```

```
- Ubuntu:
/usr/share/postgresql/17/contrib/postgis-3.5/postgis.sql
- MS Windows:
c:\PostgreSQL\17\share\contrib\postgis-3.5\postgis.sql
```

En cualquier caso, el usuario puede buscar dichos ficheros utilizando algún comando de búsqueda en el sistema de archivos o viendo los ficheros instalados por el gestor de paquetes utilizado (*synaptic package manager* en *Ubuntu* o *yast* en *OpenSuse*).

2. Crear la base de datos, por ejemplo, vamos a llamarle *s0*.

```
consola> createdb -U postgres s0
```

Si el usuario de PostgreSQL tiene el mismo nombre que el usuario que ha abierto el terminal del sistema operativo no sería necesario utilizar la opción `'-U'`. En tal caso la sentencia anterior sería similar a: `createdb s0`

La información textual (campos tipo `varchar`, etc.) se almacena en la base de datos utilizando la codificación por defecto de PostgreSQL (generalmente UTF8). También se puede elegir la codificación con la opción `-E` (p. ej.: `createdb -U postgres -E UTF8 s0`) al crear la base de datos (ver el apartado I 1.4, pág. 511 para más información).

3. Si la versión de PostgreSQL es la 9.0 o superior entonces salta al apartado 4, sino procederemos a instalar el lenguaje *PL/PgSQL* de PostgreSQL en la base de datos:

```
consola> psql -U postgres -d s0 -c "create extension plpgsql"
```

Si la base de datos ya dispone de dicho lenguaje el sistema dará un aviso: `"ERROR: la extensión «plpgsql» ya existe"`, que no afecta al proceso de instalación.

4. Ejecutar el fichero *postgis.sql* (incluido en la distribución de PostGIS) en la base de datos *s0*. Este fichero añade toda la funcionalidad espacial de PostGIS a la base de datos.

```
consola> psql -U postgres -f ruta_a_postgis.sql -d s0
```

Ejemplo con instalación del capítulo anterior (si la ruta contuviera espacios la pondríamos entre comillas):

```
psql -U postgres -f c:\PostgreSQL\13\share\contrib\postgis-3.1\postgis.sql -d s0
```

5. Ejecutar el fichero *spatial_ref_sys.sql* (incluido en la distribución de PostGIS) en la base de datos *s0*, este fichero crea una tabla con información sobre unos 4000 sistemas de referencia distintos.

```
consola> psql -U postgres -f ruta_a_spatial_ref_sys.sql -d s0
```

6. Este paso es opcional y sirve para añadir pequeños comentarios de ayuda sobre las instrucciones de PostGIS. Si se quieren añadir estos comentarios será necesario ejecutar el fichero *postgis_comments.sql* en la base de datos.

```
consola> psql -U postgres -f ruta_a_postgis_comments.sql -d s0
```

La opción `'-d'` utilizada en los comandos para indicar la base de datos es opcional y se puede omitir.

Módulo de *raster*

Además del módulo principal y vectorial (*postgis.sql*), PostGIS incluye dos bloques extra de nuevas funcionalidades dentro del núcleo del producto. Nos referimos a la gestión de datos *raster* y al modelo de topología persistente.

Ambas funcionalidades se deben añadir a nuestra base de datos ejecutando sendos ficheros SQL de la misma forma que se ha realizado hasta ahora.

Los ficheros a ejecutar en la base de datos serán: *rtpostgis.sql* y *raster_comments.sql* (si se desea incluir los comentarios de las funciones).

```
consola> psql -U postgres -f ruta_a_rtpostgis.sql -d s0
consola> psql -U postgres -f ruta_a_raster_comments.sql -d s0
```

Módulo de topología persistente

La funcionalidad correspondiente al modelo de topología persistente es opcional. Esta funcionalidad se debe añadir a la base de datos ejecutando varios ficheros SQL de la misma forma que se ha realizado hasta ahora.

Si no se va a utilizar la funcionalidad de topología persistente en PostGIS, no sería necesario realizar este paso ahora. Se puede ejecutar más adelante incluso con nuestra base de datos espacial ya cargada con cartografía.

Los ficheros SQL correspondientes al modelo de topología están en una subcarpeta llamada *topology* en la misma ubicación que los demás ficheros SQL utilizados hasta ahora.

Los ficheros a ejecutar en la base de datos serán: *topology.sql* y *topology_comments.sql* (si se desea incluir los comentarios de las funciones).

```
consola> psql -U postgres -f ruta_a_topology.sql -d s0
consola> psql -U postgres -f ruta_a_topology_comments.sql -d s0
```

Otros módulos

Existen otros módulos como *SFCGAL*, que incluye algunas funciones espaciales para hacer análisis espacial vectorial en tres dimensiones, o *pgrouting*, que añade la funcionalidad de análisis de rutas. Éstos y cualquier otro módulo extra de PostGIS también se pueden añadir a la base de datos ejecutando los correspondientes ficheros SQL, aunque no insistimos más, ya que la forma habitual es utilizar el comando *Create Extension* que veremos a continuación.

Nota sobre los ficheros SQL de PostGIS

Es interesante para el lector que eche un vistazo con un editor de texto al contenido de los ficheros SQL que PostGIS necesita cargar en la base de datos para convertirla en espacial. Como práctica se insta al lector a visualizar el contenido del fichero *postgis.sql*.

El fichero *postgis.sql* añade las funcionalidades del motor principal de PostGIS y todo el tratamiento vectorial. Estas funciones en PostGIS están desarrolladas en su mayoría en C y algunas en *PL/PgSQL*.

Si las funciones están desarrolladas en *PL/PgSQL* el propio código fuente de la función estará contenido en el fichero *postgis.sql*, por el contrario, si están desarrolladas utilizando C, el código binario de la función estará en una biblioteca (*postgis.so* en *Linux* o *postgis.dll* en *MS Windows*) y en el fichero *postgis.sql* se indicará dónde está la biblioteca y como se llama la función C dentro de ella.

Por ejemplo, si localizamos la función *ST_Intersection* en el fichero *postgis.sql* encontraremos un código similar al siguiente (instalación en *MS Windows*):

```
CREATE OR REPLACE FUNCTION ST_Intersection(geom1 geometry, geom2 geometry)
  RETURNS geometry
  AS '$libdir/postgis-3', 'ST_Intersection'
  LANGUAGE 'c' IMMUTABLE STRICT PARALLEL SAFE COST 5000;
```

Dicho código crea una nueva función SQL llamada *ST_Intersection* que a su vez llama a la función de nombre *ST_Intersection* albergada en la biblioteca que está dentro del directorio *\$libdir* (p. ej.: *c:\postgresql\17\lib*) y con nombre *postgis-3.dll* (*.so* en *SO Linux*).

En cambio, si el lector localiza la función *find_srid*, se encontrará con el código fuente *PL/PgSQL* incrustado dentro de la sentencia *CREATE OR REPLACE FUNCTION*.

Además de nuevas funciones SQL, en el fichero *postgis.sql* se define las tablas *spatial_ref_sys*, *geometry_columns*, los nuevos tipos de datos *geometry* y *geography*, conversiones explícitas, agregados, etc.

El fichero *spatial_ref_sys.sql* que se ejecuta a continuación del fichero *postgis.sql* solo rellena la tabla *spatial_ref_sys* con información de los CRS que entiende PostGIS.

La conclusión principal de este punto es que una base de datos espacial **es dependiente de la instalación de PostGIS** y del sistema operativo en que se haya realizado, ya que en la definición de las funciones **aparece la ruta** donde se encuentran las bibliotecas de PostGIS. Esto hay que tenerlo muy en cuenta a la hora de realizar un *backup* de la base de datos espacial para cargarla en otro ordenador como veremos más adelante.

De forma similar los ficheros *rtpostgis.sql* y *topology.sql* (éste último principalmente cuenta con funciones desarrolladas en *PL/PgSQL*), hacen dependiente la base de datos del ordenador en la cual esté instalado PostgreSQL / PostGIS.

1.2. Creación base de datos espacial utilizando *extensions*

A partir de la versión 9.1, PostgreSQL añade el comando SQL *Create Extension* que permite crear una base de datos espacial de una forma más sencilla y rápida no siendo necesario ejecutar ningún fichero SQL tipo *postgis.sql*, *spatial_ref_sys.sql*, *rtpostgis.sql*, etc. como se ha realizado anteriormente.

Desde hace años PostGIS aprovecha esta nueva característica de PostgreSQL para la instalación de extensiones de forma que el proceso de creación de una nueva base de datos espacial queda resumido a ejecutar el comando *Create extension* en una nueva base de datos.

```
--Borrar la base de datos anterior s0 del apartado anterior.
consola> dropdb -U postgres s0
--Crea una nueva base de datos
consola> createdb -U postgres s0
```

```
--Realiza una conexión a la base de datos
consola> psql -U postgres s0
--Instala la funcionalidad vectorial
s0=# create extension postgis;

--Instala la funcionalidad raster. Si no se va a utilizar datos raster se
puede dejar sin instalar esta extensión. Para más información
consultar el capítulo G donde se desarrolla con detalle esta extensión
de postgis.
s0=# create extension postgis_raster;

--Instala la funcionalidad de topología persistente. En la gran mayoría
de ocasiones no necesitaremos esta funcionalidad y no haría falta
instalar esta extensión. Para más información consultar el capítulo H
2, pág. 470 donde se desarrolla con detalle esta funcionalidad.
s0=# create extension postgis_topology;
```

1.3. Comprobación de la base de datos espacial

La función de PostGIS, `postgis_full_version` devuelve la versión de PostGIS, así como las versiones de las bibliotecas instaladas. Es una buena forma de comprobar que todo el proceso se ha realizado de forma correcta, independientemente de la forma elegida para crear la base de datos espacial.

```
consola> psql -U postgres s0
s0=# select postgis_full_version();
```

```
postgis_full_version
```

```
-----
POSTGIS="3.5.0 3.5.0" [EXTENSION] PGSQL="170" GEOS="3.13.0-CAPI-1.19.0" PROJ="8.2.1 NETWORK_ENABLED=OFF URL_ENDPOINT=USER WRITABLE DIRECTORY=C:\WINDOWS\ServiceProfiles\NetworkService\AppData\Local\proj" (compiled against PROJ 8.13.0) GDAL="GDAL 3.9.2, released 2024/08/13 GDAL_DATA not found" LIBXML="2.12.5" LIBJSON="0.12" LIBPROTOBUF="1.2.1" WAGYU="0.5.0 (Internal)" TOPOLOGY RASTER
```

Si se ha instalado la funcionalidad `raster`, aparecerá la palabra `RASTER` al final del texto. De la misma forma, aparecerá `TOPOLOGY`, si ha instalado la funcionalidad de topología persistente.

Para comprobar que la fecha de la compilación se puede utilizar el comando:

```
consola> psql -U postgres s0
s1=# SELECT postgis_lib_build_date();
postgis_lib_build_date
-----
2024-09-25 19:49:48
```

También se puede consultar la compilación y la versión de la funcionalidad `raster` mediante:

```
consola> psql -U postgres s0
s1=# SELECT postgis_raster_lib_build_date(),postgis_raster_lib_version();
postgis_raster_lib_build_date | postgis_raster_lib_version
-----+-----
2024-09-25 19:49:48          | 3.5.0 3.5.0
```

1.4. Funciones extra y plantillas

Como se comenta en el apartado F 3, pág. 326, el comportamiento de algunos métodos SQL de PostGIS pueden cambiar entre diferentes versiones. Estos cambios pueden provocar que ciertas sentencias SQL o *scripts* realizados dejen de funcionar entre versiones. Desde la página web de PostGIS estos cambios se documentan de forma exhaustiva¹ y se hace necesario consultarlos antes de realizar una migración de los procedimientos almacenados (apartado E de programación) o rutinas SQL que puede haber realizado el usuario.

Especialmente hay que tener cuidado con el comportamiento de los métodos *ST_StartPoint*, *ST_EndPoint* con las geometrías de tipo *Multi*. Por esta razón, para que el lector pueda ejecutar los ejemplos de esta publicación con diferentes versiones de PostGIS es aconsejable ejecutar el fichero *funcionesextra.sql* en nuestra base de datos espacial, de esta forma se agregan algunas funciones como *STX_StartPoint*, *STX_EndPoint*, *STX_Extract* y *STX_Last* que son utilizadas con frecuencia en los ejemplos de esta publicación.

```
consola> psql -U postgres -f ruta_al_fichero_funcionesextra.sql s0
```

Creación de una plantilla espacial

Como se puede apreciar el proceso de creación de una base de datos espacial es un poco tedioso y sobre todo repetitivo. Para facilitar esta labor se puede utilizar la característica que tiene PostgreSQL de las plantillas. De esta forma por ejemplo vamos a utilizar la base de datos *s0* que acabamos de crear como si fuera una plantilla para la creación de otras bases de datos espaciales. Eso sí, es aconsejable no crear ninguna tabla, ni ningún otro objeto en la base de datos *s0* posteriormente.

Para crear una nueva base de datos espacial *s1* que inicialmente sea una copia de *s0*, es algo tan sencillo como:

```
consola> createdb -U postgres -T s0 s1
```

La base de datos *s1*, será una copia de *s0*, por lo tanto, será espacial y además también contendrá las funciones extra instaladas en el apartado anterior.

PostGIS se instala en el esquema por defecto de PostgreSQL, que a no ser que se haya cambiado la configuración por defecto de la base de datos (variable *search_path*) es *public*. Es posible, aunque quizás no aconsejable, instalar PostGIS en otro esquema que no sea el *public*. Para no complicar más en esta publicación seguiremos con PostGIS instalado en el esquema *public*. El apartado F 6.2, pág. 338, detalla como instalarlo en otro esquema.

Si se va a crear una base de datos con un usuario diferente al que creó la base de datos correspondiente a la plantilla, es necesario decirle a PostgreSQL que la base de datos utilizada como plantilla, este caso *s0*, es realmente una plantilla. Esto se hace de la siguiente forma:

```
s0=# UPDATE pg_database SET datistemplate = TRUE WHERE datname = 's0';
```

En este ejemplo no es necesario ejecutar la sentencia SQL anterior, ya que la base de datos *s1* es creada por el usuario *postgres* que es el mismo usuario que creó la base de datos *s0* utilizada como plantilla.

¹ https://postgis.net/docs/release_notes.html

1.5. Metadatos sobre los CRS.

La tabla *spatial_ref_sys* contiene información descriptiva sobre los sistemas de referencia espacial o también llamados sistemas de referencia de coordenadas soportados por PostGIS. En esta publicación nos referimos a ellos mediante las siglas CRS (*Coordinate Reference System*) o simplemente como ‘sistemas de referencia’.

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

- *SRID*: Identificador único de cada CRS. Éste es el valor utilizado por PostGIS.
- *AUTH_NAME*: Nombre del estándar utilizado. Ejemplo: EPSG.
- *AUTH_SRID*: Identificador del CRS según el estándar utilizado.
- *SRTEXT*: Representación del CRS según el formato WKT.
- *PROJ4TEXT*: Representación del CRS según el formato utilizado en *PROJ4* (biblioteca de transformación de coordenadas que utiliza PostGIS).

En la versión 3.4 de PostGIS, esta tabla tiene sobre 8500 CRS distintos (registros). En casi todos ellos se ha utilizado el estándar EPSG. Actualmente, el valor del campo *SRID* coincide con el valor del campo *AUTH_SRID*.

```
s1=# select srtext from spatial_ref_sys where srid=25830;
PROJCS["ETRS89 / UTM zone 30N",
  GEOGCS["ETRS89",
    DATUM["European Terrestrial Reference System 1989",
      SPHEROID["GRS 1980",6378137,298.257222101,
        AUTHORITY["EPSG","7019"]],
      TOWGS84[0,0,0,0,0,0,0], AUTHORITY["EPSG","6258"]
    ],
    PRIMEM["Greenwich",0,AUTHORITY["EPSG","8901"]],
    UNIT["degree",0.0174532925199433,AUTHORITY["EPSG","9122"]],
    AUTHORITY["EPSG","4258"]
  ],
  PROJECTION["Transverse_Mercator"],
  PARAMETER["latitude_of_origin",0], PARAMETER["central_meridian",-3],
  PARAMETER["scale_factor",0.9996], PARAMETER["false_easting",500000],
  PARAMETER["false_northing",0],
  UNIT["metre",1,AUTHORITY["EPSG","9001"]],
  AXIS["Easting",EAST], AXIS["Northing",NORTH],
  AUTHORITY["EPSG","25830"] ]
```

El valor *SRID* = 0 indica a PostGIS que los datos no presentan ningún sistema de referencia. Este es el valor por defecto cuando se crea una geometría sin definir el *SRID*.

En el documento del *OGC* “*OpenGIS Implementation Specification: Coordinate Transformation Services*”¹ aparece la definición de un CRS mediante WKT.

La tabla *spatial_ref_sys* de PostGIS describe mediante este estándar (WKT) los CRS establecidos por diferentes organizaciones como, por ejemplo, el estándar EPSG².

¹ <https://www.ogc.org/standards/ct>

² European Petroleum Survey Group (EPSG). <https://epsg.org/home.html>

Además del estándar WKT, PostGIS necesita definir los CRS para que los entienda la biblioteca que utiliza para re proyectar, es decir, la biblioteca *proj4*. Ésta no entiende el estándar WKT y utiliza sus propios parámetros. Dichos parámetros se almacenan en la columna *PROJ4TEXT*.

```
s1=# select proj4text from spatial_ref_sys where srid=25830;
      proj4text
-----
+proj=utm +zone=30 +ellps=GRS80 +towgs84=0,0,0,0,0,0 +units=m +no_defs
```

El usuario puede editar directamente la tabla *spatial_ref_sys* con el fin de añadir un CRS nuevo o editar alguno existente. En los apartados C 8.3, p. 179 y D 2, pág. 214, se realizan ejercicios sobre proyecciones.

1.6. Creación y borrado de una tabla espacial

La creación de una tabla espacial para almacenar geometrías es similar a la creación de una tabla normal, salvo en que hay que definir un campo donde se almacenan las geometrías. Antes de pasar a ver con detalle cada uno de los tipos de geometría principales que soporta PostGIS, vamos a crear una tabla espacial para almacenar geometrías de tipo punto en 3D utilizando el CRS de código EPSG: 25830.

Al introducir una nueva geometría en una tabla, PostGIS realiza una serie de comprobaciones para admitir únicamente el tipo de geometría, el sistema de referencia y la dimensión de coordenadas (2D, 3D o 4D) que el usuario desea. Este tipo de comprobaciones se puede realizar de dos formas distintas:

- La tradicional, que consiste en aplicar **restricciones de tipo CHECK** sobre el campo de geometría.
- Una segunda forma, más actual (disponible a partir de PostGIS 2.0), se basa en la **característica *typmod*** de PostgreSQL, y que simplifica sobre todo la forma de definir dichas condiciones.

Desde un punto de vista práctico, lo más rápido y habitual es utilizar el método *typmod*. Desde un punto de vista funcional, el método *typmod* tiene algunas limitaciones como en cierto uso en los disparadores, herencia de tablas o gestión de las restricciones, aunque no vemos adecuado su explicación en estos momentos y tampoco nos va a afectar en nada a lo largo de la publicación.

Utilizando *typmod*

A su vez, la creación o adición de campos de geometría en una tabla mediante la característica *typmod* se puede realizar mediante dos gramáticas diferentes:

- La forma rápida, definiendo directamente la geometría y sus restricciones al mismo tiempo que se define el nuevo campo en una tabla. Forma que además es la más utilizada.
- La forma lenta, que respeta los estándares que consiste en utilizar dos métodos: *addgeometrycolumn* y *dropgeomercolumn*.

A nivel funcional las dos formas son exactamente iguales.

Método 1: Forma rápida

El primer paso consiste en crear una tabla convencional con los campos de atributos deseados.

```
s1=# create table ciudades (gid serial PRIMARY KEY, nombre varchar,
    poblacion integer);
```

A continuación, se añade la columna de geometría con sus parámetros. En este caso la columna de geometría será de tipo punto, capaz de almacenar XYZ y con un CRS de código EPSG: 25830.

```
s1=# alter table ciudades add column geom geometry (PointZ, 25830);
```

Es evidente que los dos pasos anteriores se pueden resumir en un único paso:

```
create table ciudades (gid serial PRIMARY KEY, nombre varchar,
    poblacion integer, geom geometry (POINTZ, 25830));
```

El parámetro del tipo de geometría especificado en el tipo *geometry* puede ser (no es sensible a mayúsculas o minúsculas): *Point*, *Multipoint*, *Linestring*, *Multilinestring*, *Polygon*, *Multipolygon*, *GeometryCollection*, más el mismo conjunto de nombres de geometrías, pero acabado en Z (p. ej.: *PolygonZ*), acabado en M (p. ej.: *MultilinestringM*) y acabado en ZM (p. ej.: *GeometrycollectionZM*). Los tipos curvos también están soportados (en los apartados B 2, pág. 51 y B 4, pág. 75, aparece una descripción completa de los diferentes tipos de geometría soportados por PostGIS).

Esta forma de configurar los parámetros de un tipo de dato, en este caso el tipo *geometry* (tipo de geometría, SRID del CRS y dimensión) es una característica introducida en PostgreSQL 8.3, se denomina *typmod* y PostGIS la aplica a partir de la versión 2.0.

La definición de la tabla (comando ‘\d’ del cliente *psql*) queda:

```
s1=# \d ciudades
          Table "public.ciudades"
  Columna |          Type          | Nullable |          Default
-----+-----+-----+-----
gid       | integer                | not null | nextval('ciudades_gid_seq..')
nombre    | character varying     |         |
poblacion | integer                |         |
geom      | geometry(PointZ,25830) |         |
Indexes:
    "ciudades_pkey" PRIMARY KEY, btree (gid)
```

Para borrar una columna de geometría de una tabla simplemente utilizaremos el comando *alter table*:

```
s1=# alter table ciudades drop column geom;
```

o si lo que se desea es borrar la tabla entera incluyendo todas las columnas de geometría:

```
s1=# drop table ciudades;
```

Método 2: Forma lenta

El primer paso es crear una tabla convencional con los campos de atributos que se desee, pero sin especificar la columna de geometría:

```
s1=# create table ciudades (gid serial PRIMARY KEY, nombre varchar,
    poblacion integer);
```

El segundo paso es utilizar la función de PostGIS “*addgeometrycolumn* (<esquema>, <tabla>, <columna>, <srid>, <tipo>, <dimension>, <usar_typmod = true>)”. Esta función añade a la tabla especificada una columna para almacenar geometrías del tipo y dimensión especificadas.

```
s1=# select addgeometrycolumn ('ciudades', 'geom', 25830, 'POINT', 3, true);
```

El primer argumento de la función es el esquema, éste se puede dejar en blanco o incluso eliminar, con lo cual PostGIS utilizará el esquema por defecto *public*.

El nombre del tipo de geometría especificado en el argumento de *Addgeometrycolumn* debe ir en mayúsculas (este argumento no es sensible al tipo de letra) y son¹: POINT, MULTIPOINT, POLYGON, MULTIPOLYGON, LINestring, MULTILINESTRING, GEOMETRYCOLLECTION.

El último parámetro *usar_typmod* es opcional y toma por defecto el valor *true*. Si su valor es *false* entonces se crearán restricciones de tipo *check* y no se utilizará la característica *typmod*, caso que veremos a continuación.

El método *addgeometrycolumn* es un método propuesto por el OGC para añadir una columna espacial a una tabla, aunque el último argumento no forma parte del estándar.

La tabla creada tiene exactamente el mismo aspecto que la de *ciudades* listada anteriormente, por tanto, omitimos la salida de `\d ciudades`. La única ventaja es respetar mejor los estándares que aconsejan utilizar el método *addgeometrycolumn* para añadir columnas de geometría a una tabla.

Para borrar una columna de geometría podemos igual que en el apartado anterior utilizar una sentencia *alter table*, o podemos seguir respetando los estándares y utilizar el método *dropgeometrycolumn*.

Borra una columna de geometría de una tabla:

```
s1=# select dropgeometrycolumn ('ciudades', 'geom');
```

Borra una tabla:

```
s1=# select dropgeometrytable ('ciudades');
```

Las funciones *Addgeometrycolumn*, *Dropgeometrycolumn* y *Dropgeometrytable* tienen variaciones que toman como primer argumento el esquema de la tabla. Por ejemplo, si la tabla está situada en el esquema *ej1* la sintaxis sería: *addgeometrycolumn ('ej1', 'ciudades', 'geom', 25830, 'POINT', 3)*;

¹ PostGIS dispone de otros tipos de geometría para manejar objetos compuestos y/o curvos, aunque muchos de los comandos actuales aún no soportan estos tipos. Ver el apartado F 8 para más información.

Utilizando restricciones de tipo *check*

Si queremos crear o añadir un campo de geometría a una tabla utilizando restricciones de tipo *check*, la forma más sencilla en este caso será utilizar el método *addgeometrycolumn*.

La forma será igual que en el apartado anterior pero el último argumento *typmod* será igual a falso para utilizar restricciones de tipo *check*.

```
s1=# create table ciudades (gid serial PRIMARY KEY, nombre varchar,
    poblacion integer);

s1=# select addgeometrycolumn ('ciudades', 'geom', 25830, 'POINT', 3, false);
```

La definición de la tabla (comando ‘\d’ del cliente *psql*) quedará:

```
s1=# \d ciudades
```

Column	Type	Nullable	Default
gid	integer	not null	nextval('ciudades_gid_seq...)
nombre	character varying		
poblacion	integer		
geom	geometry		

Indexes:

```
"ciudades_pkey" PRIMARY KEY, btree (gid)
```

Check constraints:

```
"enforce_dims_geom" CHECK (st_ndims(geom) = 3)
"enforce_geotype_geom" CHECK (geometrytype(geom) = 'POINT' OR geom IS NULL)
"enforce_srid_geom" CHECK (st_srid(geom) = 25830)
```

Como se puede apreciar los parámetros de la geometría especificados (dimensión de las coordenadas, sistema de referencia y tipo de geometría) se implementan mediante tres restricciones de tipo *Check*. Dentro de estas restricciones se puede ver tres comandos de PostGIS: *ST_Ndims*, *geometrytype* y *ST_Srid* que devuelven la información requerida de la geometría.

Por lo tanto, la orden *Addgeometrycolumn* además de ejecutar la correspondiente orden *Alter* para añadir la columna de geometría ha añadido tres nuevas restricciones de tipo *check* a la tabla.

Para borrar una columna de geometría de una tabla o la tabla entera, se procede igual que anteriormente:

```
s1=# alter table ciudades drop column geom;
s1=# drop table ciudades;
```

Al borrar la columna también se borrarán las restricciones de tipo *check* automáticamente, aunque si queremos estar seguros y además respetar mejor los estándares podemos utilizar los métodos *dropgeometrycolumn* o *dropgeometrytable* como hemos mostrado anteriormente.



Problema 1. Crea una tabla *capatest* con una columna de geometría llamada *shape* de tipo *LineString*, con coordenadas XYZM y un CRS 25830. Utiliza los dos métodos explicados: es decir, creando restricciones *check* y utilizando la característica *typmod*. Tras crear las tablas elimínalas de forma adecuada.

1.7. Metadatos de las columnas de geometría

En PostGIS existe una **vista** de metadatos que inventaría la información descriptiva sobre las columnas de geometría de todas las tablas que existen en la base de datos. Se denomina *geometry_columns* y tiene el mismo número y tipo de campos en todas las versiones de PostGIS.

Al ser *geometry_columns* una vista, que proviene de una consulta del catálogo de PostgreSQL, su contenido se actualiza de forma automática, y por tanto se evitan los problemas que tenían las versiones antiguas de PostGIS de falta de sincronización.

```
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
)
```

- *F_TABLE_CATALOG*: Nombre del catálogo (en PostGIS se corresponde con el nombre de la base de datos).
- *F_TABLE_SCHEMA*: Nombre del esquema al que pertenece la tabla. El esquema es por defecto *public*.
- *F_TABLE_NAME*: Nombre de la tabla que contiene la columna de geometría.
- *F_GEOMETRY_COLUMN*: Nombre de la columna de geometría.
- *COORD_DIMENSION*: Dimensión espacial (2, 3 ó 4) de las coordenadas de la columna de geometría.
- *SRID*: Identificador del CRS (es una clave ajena de la tabla *spatial_ref_sys*).
- *TYPE*: El tipo de geometría de la columna. Este campo de la tabla no es obligatorio según el OGC, pero PostGIS lo añade para mantener la homogeneidad de geometrías dentro de una tabla.

```
s1=# select * from geometry_columns;
 f_table_catalog | f_table_schema | f_table_name | f_geometry_column |
 coord_dimension | srid          |              | type               |
-----+-----+-----+-----+-----+-----+-----+-----+
      s1         | public        | ciudades     | geom               |
                | 3            |              | POINT              |
```



Problema 2. Averigua si PostGIS permite añadir varias columnas de geometría en la misma tabla. ¿Encuentras algún significado o aplicación práctica al que una tabla disponga de más de una columna de geometría?

2. Tipos de geometría

En los estándares comentados en el apartado A 1.1, pág. 24, se definen diferentes tipos de geometrías. Los tipos de geometrías que PostGIS soporta con plena funcionalidad son principalmente los expuestos en los estándares SFS1.1 / SFA1.1.0, es decir, los tipos *Point*, *Multipoint*, *Linestring*, *Multilinestring*, *Polygon*, *Multipolygon* y *Geometrycollection*. Estos tipos han sido soportados por PostGIS desde su inicio y además son los tipos básicos en los que se apoya cualquier SIG de escritorio.

Además, en dichos estándares se definen dos formas de representación espacial de las geometrías: WKT (*Well-Known Text*) y WKB (*Well-Known Binary*). Algunos ejemplos de representación WKT de estos siete tipos de geometrías son:

Tipo de Geometría	Representación WKT	Comentario
<i>POINT</i>	“POINT (2 2)”	un <i>Point</i>
<i>LINESTRING</i>	“LINESTRING (4 1, 1 4, 1 1, 4 3)”	un <i>Linestring</i> con 3 <i>Point</i>
<i>POLYGON</i>	“POLYGON ((1 4, 1 1, 4 1, 4 2, 3 2, 3 4, 1 4))” “POLYGON ((1 1, 5 1, 5 5, 1 5, 1 1), (2 2, 4 2, 4 4, 2 4, 2 2))”	un <i>Polygon</i> con 1 <i>exteriorRing</i> y 0 <i>interiorRing</i> (isla) un <i>Polygon</i> con 1 <i>exteriorRing</i> y 1 <i>interiorRing</i> (isla)
<i>MULTIPOINT</i>	“MULTIPOINT (1 1, 2 4, 3 2)” o “MULTIPOINT ((1 1), (2 4), (3 2))”	un <i>MultiPoint</i> con 3 <i>Point</i> . Ambas sintaxis son aceptadas por PostGIS.
<i>MULTILINESTRING</i>	“MULTILINESTRING ((1 4, 1 1, 3 1), (4 5, 2 2, 4 2, 2 4))”	un <i>MultiLinestring</i> con 2 <i>linestrings</i>
<i>MULTIPOLYGON</i>	“MULTIPOLYGON (((1 4,4 4,4 1,1 1,1 4), (3 2,2 2,2 3,3 3,3 2)), ((1 6, 4 6, 4 5, 1 5, 1 6)))”	un <i>MultiPolygon</i> con 2 <i>polygon</i> (1 de ellos con una isla)
<i>GEOMETRY COLLECTION</i>	“GEOMETRYCOLLECTION (LINESTRING (1 4, 2 4, 2 2), LINESTRING (1 3, 1 1, 2 1), POINT (3 4), POINT (3 1), POLYGON ((4 4, 4 1, 7 1, 7 4, 4 4), (5 3, 5 2, 6 2, 5 3)), POLYGON ((8 4, 8 1, 9 1, 9 4, 8 4)))”	una <i>GeometryCollection</i> formada por 2 <i>Point</i> , 2 <i>Linestring</i> y 2 <i>Polygon</i> . También puede estar formada por elementos de tipo <i>Multi</i> así como de otras <i>GeometryCollection</i> incluso de forma anidada.

Tabla 4 Representación WKT de objetos espaciales

En la figura siguiente se muestra de forma gráfica la representación de estos siete tipos de geometrías.

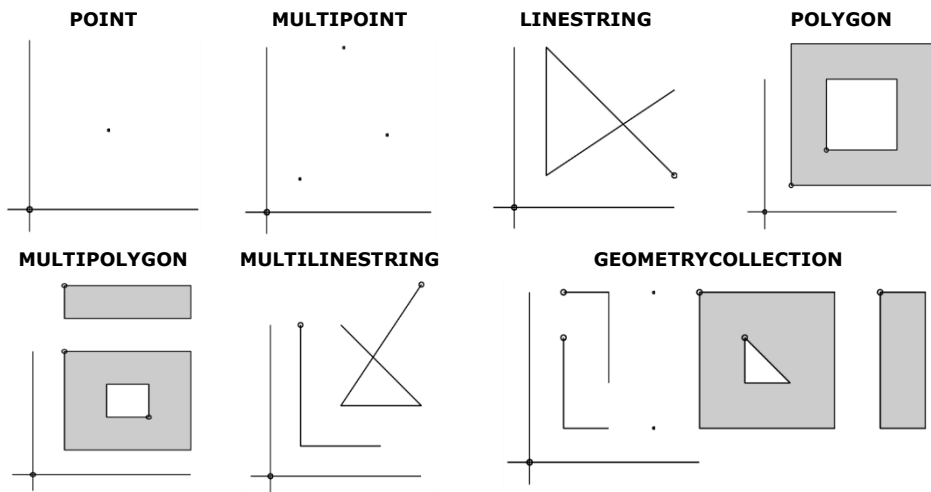


Figura 7 Tipos básicos de geometrías en PostGIS

Posteriormente y siguiendo la evolución de las normas SFA y SQL/MM, PostGIS ha ido incluyendo otros tipos de geometrías como curvas circulares o superficies TIN. En el apartado B 4, pág. 75, se da una visión general de todos los tipos de geometrías de PostGIS, así como del esquema de herencia de las geometrías seguido por PostGIS.

Vértices con Z, M o ZM

PostGIS permite representar el WKT de geometrías con coordenadas Z, M o ZM de dos formas diferentes: con sufijo Z, M, ZM o sin él. A continuación, se muestra el formato por defecto de una geometría de tipo punto (es extrapolable a cualquier tipo de geometría):

Descripción	WKT soportado por PostGIS (geometría puntual)
Punto 3D	POINTZ (2 2 3), POINT Z (2 2 3) o POINT (2 2 3)
Punto 3D con ZM	POINTZM (2 2 3 4), POINT ZM (2 2 3 4) o POINT (2 2 3 4)
Punto 2D con M	POINTM (2 2 4) o POINT M (2 2 4)

Tabla 5 Representación WKT de una geometría puntual según las coordenadas Z, M o ZM

2.1. Creación e inserción de geometrías

Para construir una geometría en PostGIS se puede utilizar la definición en texto (WKT) o la definición en binario (WKB) de las mismas. Para ello, se dispone de los constructores *ST_Geomfromtext* (*geometry, srid*) y *ST_Geomfromwkb* (*geometry, srid*) respectivamente.

- Los constructores de geometría cogen la geometría de su argumento (en formato WKT o WKB) y devuelven un bloque binario (BLOB) listo para poder ser almacenado en la columna de geometría de la tabla correspondiente.
- Los lectores de geometría cogen la geometría de su argumento (en formato binario BLOB) y la devuelven en formato WKT o WKB.

El formato BLOB o *geometry* interno de PostGIS no tiene por qué coincidir con el formato WKB / EWKB, además puede variar con las diferentes versiones de PostGIS. Por ello, cuando el usuario quiere acceder de forma binaria a las geometrías siempre ha de utilizar los lectores *ST_AsBinary* o *ST_AsEwkb*, ya que el formato WKB/EWKB está ampliamente documentado y no debe variar en futuras versiones.

En la siguiente tabla se muestran los constructores y lectores de geometrías en formato texto (WKT) y en binario (WKB) junto con sus versiones extendidas EWKT/EWKB.

Formato	Constructor	Lector
WKT	-BLOB <i>st_geomfromtext</i> (WKT, SRID) -BLOB <i>st_geomfromtext</i> (WKT) SRID por defecto = 0 Alias: <i>st_geometryfromtext</i>	-WKT <i>st_astext</i> (BLOB) No incluye el SRID dentro de la cadena de texto de la geometría.
WKB	-BLOB <i>st_geomfromwkb</i> (WKB, SRID) -BLOB <i>st_geomfromwkb</i> (WKB) SRID por defecto = 0	-WKB <i>st_asbinary</i> (BLOB) -WKB <i>st_asbinary</i> (BLOB, encoding) No incluye el SRID dentro de la cadena de texto de la geometría.
EWKT	-BLOB <i>st_geomfromewkt</i> (EWKT)	-EWKT <i>st_asewkt</i> (BLOB) Este lector no es compatible con los estándares porque incrusta el SRID dentro de la cadena de texto de la geometría.
EWKB	-BLOB <i>st_geomfromewkb</i> (EWKB)	-EWKB <i>st_asewkb</i> (BLOB) -EWKB <i>st_asewkb</i> (BLOB, encoding) Este lector no es compatible con los estándares porque incrusta el SRID dentro de la cadena de texto de la geometría.
TWKB	En PostGIS 2.2 se incluyó un nuevo formato binario (no estándar) llamado TWKB (<i>Tiny Well known Binary</i>) que disminuye el número de bytes necesarios de forma notable para codificar una geometría (comparado con WKB). Para ello, gestiona el número de decimales de las geometrías y realiza una compresión basada en enteros de longitud variable ¹ . Su utilización (mediante el lector <i>ST_AsTWKB</i>) está enfocada principalmente a mejorar los tiempos de transporte de la red en una arquitectura cliente-servidor (p. ej.: en visores online conectados mediante red a un servidor PostGIS).	
Todos	Todos los formatos soportan coordenadas XY, XYZ, XYZM, XYM	

Tabla 6 Constructores y lectores de geometrías

Aunque en el manual *online* de PostGIS aparecen otros constructores, los que aparecen en la tabla son los que se suelen utilizar. Además, PostGIS cuenta con otro tipo de constructores como *ST_MakePoint* o *ST_MakePolygon* que toman argumentos diferentes a una cadena WKT como las coordenadas de los puntos en doble precisión.

La inserción de una fila en la tabla *ciudades* (creada en el apartado anterior) sería:

¹ <https://developers.google.com/protocol-buffers/docs/encoding#varints>

```
s1=# insert into ciudades (nombre, poblacion, geom) values ('Ciudad A',
10000, st_geomfromtext ('POINT(700000 4500000 100)', 25830));
```

Si se trata de introducir en la misma tabla otras geometrías en otro CRS o con diferente dimensión de coordenadas, PostGIS devolverá un error.

```
s1=# insert into ciudades (nombre, poblacion,geom) values ('Ciudad A',
10000, st_geomfromtext ('POINT(700000 4500000)', 25830));
ERROR: Column has Z dimension but geometry does not
```

```
s1=# insert into ciudades (nombre, poblacion,geom) values ('Ciudad A',
10000, st_geomfromtext ('POINT(700000 4500000 100)', 25831));
ERROR: Geometry SRID (25831) does not match column SRID (25830)
```

```
s1=# insert into ciudades (nombre,poblacion,geom) values ('Ciudad A',
10000, st_geomfromtext ('MULTIPOINT(700000 4500000 100, 700000 4600000
200)', 25830));
ERROR: Geometry type (MultiPoint) does not match column type (Point)
```

Para convertir el formato BLOB de la geometría a un formato entendible por el ser humano utilizaremos los lectores en WKT:

```
s1=# select st_astext(geom) from ciudades where nombre='Ciudad A';
st_astext
-----
POINT Z (700000 4500000 100)
```

El formato WKT no incluye el SRID de la geometría (tanto los estándares SFA como SQL/MM tampoco lo hacen), por esta razón PostGIS realiza una extensión al formato WKT denominada *WKT Extended* (EWKT).

```
s1=# select st_asewkt(geom) from ciudades where nombre='Ciudad A';
st_asewkt
-----
SRID=25830;POINT(700000 4500000 100)
```

De la misma forma se puede utilizar el constructor *ST_GeomFromEWKT* especificando el SRID dentro del texto en lugar de usar un segundo argumento.

```
s1=# insert into ciudades (nombre,poblacion,geom) values ('Ciudad B',
10000, st_geomfromewkt ('SRID=25830;POINT(710000 4500000 100)'));
```

Todo formato *WKT/WKB* es también un formato *EWKT/EWKB* válido (actualmente) aunque el contrario no es cierto.

Los lectores binarios en WKB y EWKB permiten especificar el orden de los *bytes* en el segundo argumento de las funciones. El orden puede ser 'NDR' o *little-endian* (el *byte* menos significativo va primero) o 'XDR' o *big-endian* (el *byte* más significativo va primero). Si no se especifica el argumento entonces por defecto los lectores devuelven la ristra binaria con el *byte* menos significativo primero.

```
s1=# select st_asewkb(geom,'NDR') from ciudades;
st_asewkb
-----
\x01010000a0f659000000000000c05c254100000000882a51410000000000005940
s1=# select st_asewkb(geom,'XDR') from ciudades;
st_asewkb
-----
\x00a0000001000059f641255cc00000000041512a88000000004059000000000000
```

El lector binario en TWKB está orientado a comprimir el binario WKB para mejorar los tiempos de transporte en la red, aunque modifica totalmente el estándar WKB.

```
s1=# select st_astwkb(geom) from ciudades;
      st_astwkb
```

```
-----
\x010801c0b955c0a8a504c801
```



Problema 3. Visualiza el formato binario almacenado en la tabla *ciudades* (BLOB) y su equivalencia en WKB y EWKB. ¿Qué puedes decir al respecto?

Conversiones a otros formatos

Además de los lectores WKT/EWKT/WKB/EWKT, PostGIS tiene otros lectores de geometrías que permiten convertir las geometrías de formato BLOB a formato SVG¹, GML², GeoJSON³, X3D⁴, KML⁵ y MVT⁶. Aunque estas funciones espaciales no están consideradas por el OGC pueden ser útiles para convertir datos PostGIS a otros formatos.

```
s1=# select st_assvg(geom) from ciudades;
      st_assvg
```

```
-----
cx="700000" cy="-4500000"
```

```
s1=# select st_asgml(geom) from ciudades;
```

```
st_asgml
```

```
-----
<gml:Point srsName="EPSG:25830">
```

```
<gml:coordinates>700000,4500000,100</gml:coordinates></gml:Point>
```

```
s1=# select st_asgeojson(geom) from ciudades;
```

```
st_asgeojson
```

```
-----
{"type": "Point", "coordinates": [700000, 4500000, 100]}
```

```
s1=# select st_asx3d(geom) from ciudades;
```

```
st_asx3d
```

```
-----
700000 4500000 100
```

```
s1=# select st_askml(geom) from ciudades;
```

```
st_askml
```

```
-----
<Point><coordinates> -0.635451186669998, 40.625939715823385, 100
</coordinates></Point>
```

¹ Scalable Vector Graphics (SVG) es una familia de especificaciones basadas en el formato XML para describir vectores gráficos en 2D. Es un estándar desarrollado por el W3C. <https://www.w3.org/Graphics/SVG/>

² Geography Markup Language (GML) es una gramática XML definida por el OGC para describir fenómenos geográficos. <https://www.ogc.org/standards/gml>

³ GeoJSON es un formato abierto para codificar estructuras geográficas. Está basado en JSON (*JavaScript Object Notation*). <https://geojson.org/>

⁴ X3D es un estándar ISO basado en XML para representar objetos 3D. Es el sucesor del *Virtual Reality Modeling Language* (VRML). <https://www.web3d.org/x3d/>

⁵ Keyhole Markup Language (KML) es un lenguaje basado en XML para representar fenómenos geográficos dentro de un entorno basado en Internet, soporta mapas 2D y 3D. Es un estándar del OGC. <https://www.ogc.org/standards/kml>

⁶ Mapbox Vector Tile (MVT), es un formato estándar ampliamente utilizado para teselas vectoriales <https://www.mapbox.com/vector-tiles/>

Estos lectores tienen otras variedades que toman más argumentos para especificar la precisión de coordenadas de salida, la versión del formato, etc. No todas estas conversiones soportan todos los tipos de geometrías de PostGIS (sí que soportan las geometrías básicas).

Recordatorio de funciones PostGIS

Como recordatorio, cabe mencionar que se han revisado los constructores de geometrías en los formatos *WKT*, *WKB*, *EWKT* y *EWKB*, así como los lectores correspondientes para estos mismos formatos. Además, se han presentado otros lectores útiles, como *ST_AsKML* o *ST_AsGML*, junto con algunas funciones empleadas en las restricciones de las geometrías:

```
s1=# select st_srid(geom) from ciudades;
s1=# select geometrytype(geom) from ciudades;
s1=# select st_coorddim(geom) from ciudades;
```

Conversiones automáticas al tipo *geometry*

Aunque la forma adecuada de crear objetos geométricos a partir de texto *WKT* / *EWKT* es mediante los constructores de PostGIS, a veces resulta un poco tedioso tener que escribir cada vez las órdenes *ST_Geomfromtext* o *ST_Geomfromewkt*. Por esta razón PostGIS redefine los llamados *CAST* en SQL que realizan conversiones de forma automática. En concreto, hay definidos *CAST* para conversiones del tipo *text* o *varchar* al tipo *geometry*. De esta forma, se puede utilizar los métodos PostGIS que utilizan argumentos de tipo *geometry* pasándoles directamente un texto. El uso de *CAST* automáticos puede ser confuso para un usuario recién iniciado (al confundirse y creer que PostGIS está trabajando con geometrías en formato texto directamente) y además no respeta los estándares, por ello en esta publicación trataremos de no abusar de su utilización. Ejemplos:

```
s1=# select st_coorddim ('POINT (2 3 4 5)');
      st_coorddim
-----
          4
s1=# insert into ciudades (nombre,poblacion,geom) values
      ('Ciudad B', 10000, 'SRID=25830;POINT(710000 4500000 100)');
```

Relajación de las restricciones de una columna de geometría

Las geometrías almacenadas en una columna de geometría de PostGIS deben cumplir restricciones de tipo de geometría, SRID y número de dimensiones (coordenadas).

En ocasiones puede interesar que dichas restricciones sean más flexibles y dar la posibilidad de almacenar en una misma columna diferentes tipos de geometrías, o incluso geometrías con diferentes SRID.

Para flexibilizar el tipo de geometría se utiliza como tipo de geometría el tipo genérico *GEOMETRY* (o según las dimensiones deseadas *GEOMETRYZ*, *GEOMETRYM* o *GEOMETRYZM*). De esta forma PostGIS permite almacenar cualquier tipo de geometría.



Permite que PostGIS pueda almacenar entidades de tres dimensiones (XYZ) puntuales, lineales o superficiales (polígonos) en una misma columna de geometría.

```
s1=# create table ciudades_test (gid serial PRIMARY KEY, nombre varchar,
      poblacion integer, geom geometry (GEOMETRYZ, 25830));
```


De la misma forma, si en lugar de 25830, se utilizara el valor 0 (o simplemente no se especifica), entonces PostGIS permitirá almacenar geometrías con diferente SRID:

```
s1=# create table ciudades_test (gid serial PRIMARY KEY, nombre varchar,
  poblacion integer, geom geometry (GEOMETRYZ));
```

Si utilizamos restricciones de tipo CHECK en lugar de *typmod* (con el comando *addgeometrycolumn*) también se permite esta flexibilidad, utilizando el tipo GEOMETRY como argumento de *addgeometrycolumn*. Aunque siempre se puede directamente borrar (ALTER TABLE ... DROP CONSTRAINT ...) la restricción (SRID, dimensiones o tipo de geometría) que se desee eliminar.

2.2. Importación y exportación de cartografía.

PostGIS incorpora en su distribución oficial los comandos *shp2pgsql* y *pgsql2shp* para convertir, respectivamente, capas en formato *shape* a capas PostGIS y viceversa.

Estos comandos están bastante limitados, ya que solo soportan el formato *shape*, formato que por otra parte empieza ya a estar bastante en desuso.

En este mismo apartado veremos otros métodos de importación/exportación de cartografía más potentes, como utilizar un SIG de escritorio como QGIS o las bibliotecas GDAL.

Comandos *shp2pgsql*, *pgsql2shp* y *shp2pgsql-gui*

PostGIS añade la utilidad *shp2pgsql* para convertir capas en formato *shape* a tablas PostGIS (y viceversa con *pgsql2shp*). El comando *shp2pgsql* se invoca desde la consola del sistema y lo vamos a utilizar para importar varias capas cartográficas a PostGIS.

Para ver los detalles del comando se puede consultar la ayuda *online* de PostGIS o simplemente teclear *shp2pgsql*.

Algunas opciones de este comando son:

- *-w*: Formato de salida WKT (con un fin académico utilizaremos esta opción para poder ver las geometrías en un formato comprensible, por defecto el conversor *shp2pgsql* convertirá las geometrías *shape* a binario EWKB, lo cual resulta más rápido y también más exacto).
- *-s <SRID>*: Especifica el CRS (por defecto toma el valor 0).
- *-a*: Añade geometrías a una tabla ya creada.
- *-d*: Borra la tabla antes de empezar.
- *-g <columna_geometría>*: Establece el nombre para la columna de geometría (por defecto es *the_geom*).
- *-N <política>*: Establece la política cuando encuentra geometrías nulas en el *shape* (política: *insert*, *skip*, *abort*).
- *-I*: Crea un índice espacial en la columna de geometría.
- *-W <encoding>*: Establece la codificación de los caracteres utilizados en el fichero *.dbf* asociado al formato *shape*.
- *-G*: Utiliza el tipo *geography* en lugar de *geometry*. Requiere que los datos estén en coordenadas geográficas (p. ej.: WGS84 con SRID = 4326). Véase el apartado D 5.2, pág. 235, sobre análisis espacial utilizando coordenadas geográficas.
- *-t <dimensión>*: Trunca la dimensión de las geometrías a 2D, 3DZ, 3DM o 4D.
- *-T <tablespace>*: Especifica el *table space* de la nueva tabla.

El comando *shp2pgsql* genera código SQL de PostGIS para crear la tabla espacial y añadir las geometrías importadas del *shape* mediante comandos SQL de tipo *Insert*.

La creación de esquemas facilita la organización de la base de datos (ver la sección esquemas en el apartado I 2.3, pág. 537). Además, también facilita las operaciones de copia de seguridad de la base de datos espacial (véase el apartado F 6, pág. 335). Por lo tanto, es una buena práctica empezar a familiarizarse con los esquemas en este momento.

El primer paso que vamos a realizar es la creación de un esquema donde se va a trabajar con estos primeros ejemplos de cartografía.

```
s1=# create schema ej1;
```

El siguiente comando lee el fichero *shape pusossuelo* y crea la tabla espacial *pusos*. Además, utiliza la opción *-w* para que la importación se realice utilizando WKT, la opción *-g* para cambiar el nombre de la columna de geometría a *geom* (por defecto es *the_geom*), y la opción *-s* para especificar el CRS del fichero *shape* (en este caso ETRS89 UTM Huso 30, es decir, el 25830 en código EPSG).

```
consola> shp2pgsql -s 25830 -g geom -w pusossuelo.shp ej1.pusos
```

Aunque el fichero *shape* puede contener un fichero con extensión *prj* donde se indica el CRS, el cargador *shp2pgsql* en estos momentos no es capaz de interpretar dicho fichero con lo cual es necesario utilizar la opción *-s* para indicar el CRS de una forma manual.

Si el lector no es familiar con los códigos EPSG de los CRS puede descargarse la base de datos EPSG en formato *MS Access* de la página web del *OGP Geomatics Committee*¹. Mediante formularios se puede buscar los diferentes códigos EPSG, así como conocer sus parámetros. También se puede consultar los CRS *online*² así como sus diferentes formatos. Para más información consultar el apartado C 8.2, pág. 178, donde hay ejercicios sobre CRS y proyecciones.

Se obtendrá una salida por consola con el código SQL generado similar a:

```
SET CLIENT_ENCODING TO UTF8;
SET STANDARD_CONFORMING_STRINGS TO ON;
BEGIN;
CREATE TABLE "ej1"."pusos" (gid serial, "tuso" int2);
ALTER TABLE "ej1"."pusos" ADD PRIMARY KEY (gid);
SELECT AddGeometryColumn('ej1','pusos','geom','25830','MULTIPOLYGON',2);
INSERT INTO "ej1"."pusos" ("tuso",geom) VALUES
  ('500','SRID=25830;MULTIPOLYGON(((4358.07763671875 6924.2294921875,
    4329.0146484375 6954.85498046875,
    ...
    ...
    ,4358.07763671875 6924.2294921875))))');
INSERT INTO "ej1"."pusos" ("tuso",geom) VALUES ...
...
...
COMMIT;
```

¹ <https://www.epsg.org/>

² <https://spatialreference.org>

Es interesante destacar que para introducir las geometrías no se utiliza ningún constructor como se puede ver en el código SQL. En realidad, funciona porque PostGIS redefine las conversiones (AUTO CAST SQL) cuando se convierte de un texto a un tipo *geometry*, de forma que internamente está utilizando el comando *ST_Geomfromtext*. De todas formas, quedaría más claro y sería incluso más fiable si el cargador *shp2pgsql* utilizara *ST_Geomfromtext* en lugar de dejar el trabajo a las conversiones automáticas de PostGIS.

Este código SQL es el que hay que ejecutar dentro de la base de datos, para ello vamos a re-direccionar la salida del comando a un fichero de texto para posteriormente ejecutar dicho fichero (que contiene código SQL) en la base de datos *s1*.

```
consola> shp2pgsql -s 25830 -g geom -w pusossuelo.shp ej1.pusos > pusos.sql
consola> shp2pgsql -s 25830 -g geom -w psuelos.shp ej1.psuelos > psuelos.sql
consola> shp2pgsql -s 25830 -g geom -w prios.shp ej1.prios > prios.sql
consola> shp2pgsql -s 25830 -g geom -w palcanta.shp ej1.palcanta > palcanta.sql
```

A continuación, ejecutamos los cuatro ficheros SQL en la base de datos.

```
consola> psql -U postgres -f pusos.sql s1
consola> psql -U postgres -f psuelos.sql s1
consola> psql -U postgres -f prios.sql s1
consola> psql -U postgres -f palcanta.sql s1
```

En *Linux* podemos hacer uso de las tuberías para re-direccionar la salida del comando *shp2pgsql* y utilizarla directamente como entrada en el comando *psql*. De esta forma con un único comando se pueden cargar los datos. Para ello se utiliza la tubería '|' de la siguiente forma: *shp2pgsql palcanta.shp palcanta -w -g geom | psql -d s1*

Por otra parte, la tubería '>&' nos puede servir para redirigir la salida y la entrada al mismo tiempo a un fichero de texto. Tras la ejecución dicho fichero mostrará todos los resultados junto con los comandos SQL de la entrada.

Problemas de codificación de los ficheros shape

Con frecuencia las tablas de atributos (extensión *dbf*) de los ficheros *shapes* están codificadas en un juego de caracteres diferente al usado por defecto. Si esto es así al cargar el fichero SQL generado en la base de datos se producirá un error y PostgreSQL abortará el proceso. Con la opción '-W' se puede establecer la codificación del fichero *dbf* para que *shp2pgsql* lo lea de forma correcta.

Ejemplo: *shp2pgsql -g geom -w -W LATIN1 pusossuelo.shp pusos > pusos.sql*

Algunos de los juegos de caracteres¹ soportados por PostgreSQL son los siguientes:

¹ La lista completa se puede encontrar en: <https://www.postgresql.org/docs/current/multibyte.html>

Nombre	Descripción	Lenguaje	Bytes	Alias
LATIN1	ISO 8859-1, ECMA 94	Europeo occidental	1	ISO88591
LATIN2	ISO 8859-2, ECMA 94	Europeo central	1	ISO88592
LATIN3	ISO 8859-3, ECMA 94	Sur europeo	1	ISO88593
LATIN4	ISO 8859-4, ECMA 94	Norte europeo	1	ISO88594
SQL_ASCII	Sin especificar*	Algunos	1	
UTF8	Unicode, 8-bit	Todos	1-4	Unicode
WIN1250	MS Windows CP1250	Europeo central	1	
WIN1252	MS Windows CP1252	Europeo occidental	1	
EUC_JP	Código-JP Unix extendido	Japonés	1-3	

* Los valores entre 0 y 127 son tratados de acuerdo al ASCII estándar. Los valores entre 128 y 255 no se interpretan.

Tabla 7 Juegos de caracteres soportados por PostgreSQL

Versión gráfica de *shp2pgsql*

PostGIS incluye la versión gráfica *shp2pgsql-gui* del importador de ficheros *shape* dentro de la propia instalación de PostGIS.

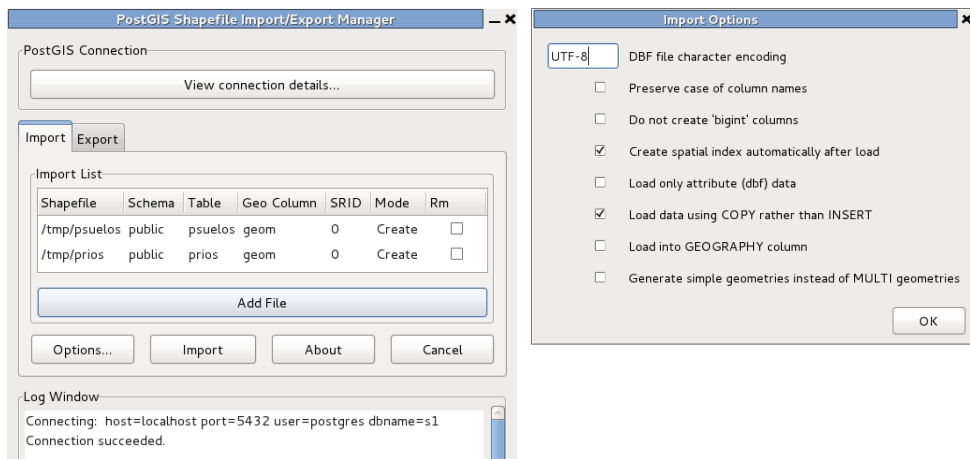


Figura 8 Versión gráfica del comando *shp2pgsql*: *shp2pgsql-gui*

En entornos *MS Windows* generalmente se encuentra en “*directorio_de_postgres\bin\postgisgui\shp2pgsql-gui.exe*”. En entornos *Linux* se suele instalar dentro del directorio de binarios del sistema y el comando se llama *shp2pgsql-gui*.

Aunque esta versión puede resultar más agradable para los usuarios noveles, en realidad es menos potente que el comando *shp2pgsql* al contar con menos opciones. Además, el comando *shp2pgsql* se puede incluir dentro de un *script* o en un proceso de lotes.